TECHNIQUES FOR AUTOMATIC DEDUCTION

Final Report

October 1981

By: Robert E. Shostak, Senior Computer Scientist
P. Michael Melliar-Smith, Senior Computer Scientist
Richard L. Schwartz, Computer Scientist

Computer Science Laboratory
Computer Science and Technology Division

Prepared for:

Air Force Office of Scientific Research
Mathematical and Information Sciences
Department of the Air Force
Bolling Air Force Base
Washington, D.C. 20332

Attention: Captain William Price

SRI Project 8752
AFOSR Contract No. F49620-79-C-0099

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
(415) 326-6200
Cable: SRI INTL MPK
TWX: 910-373-2046

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 81 -0853	2. GOVT ACCESSION NO. AD A108479	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Techniques for Automatic Deduction		5. TYPE OF REPORT & PERIOD COVERED FINAL TECHNICAL REPORT July 1979-July 1981 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert E. Shostak P.M. Meliar-Smith Richard L. Schwartz		8. CONTRACT OR GRANT NUMBER(s) F49620-79-C-0099
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Laboratory SRI International 333 Ravenswood Avenue, Menlo Park, CA 94025		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2304/A2 61102F
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB, Washington DC 20332		12. REPORT DATE October 1981 13. NUMBER OF PAGES 101
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simplification, verification, theorem proving, deduction, program correctness

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report covers progress on a 2-year research effort toward the development of new theorem-proving methods for program verification, and the empirical investigation of these methods in actual verification systems. The research conducted during the course of the project focused on methods for simplifying formulas of the kind that arise frequently in the verification of programs. The importance of simplification methods, as opposed to pure proof methods, was pointed up by verification work conducted under a previous AFOSR contract. Perhaps the most significant outcome of the project is the development of an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A108479

20. (continued)

experimental theorem prover that has been used extensively in the proof of correctness of the design of a fault-tolerant operating system developed under NASA support. We believe that the technology embodied in this experimental system could be successfully applied to the development of a production verification system.

SRI International

TECHNIQUES FOR AUTOMATIC DEDUCTION

Final Report

October 1981

By: Robert E. Shostak, Senior Computer Scientist
P. Michael Melliar-Smith, Senior Computer Scientist
Richard L. Schwartz, Computer Scientist

Computer Science Laboratory
Computer Science and Technology Division

Prepared for:

Air Force Office of Scientific Research
Mathematical and Information Sciences
Department of the Air Force
Bolling Air Force Base
Washington, D.C. 20332

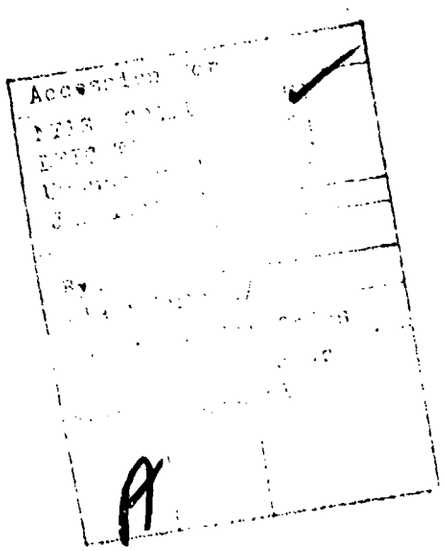
Attention: Captain William Price

SRI Project 8752
AFOSR Contract No. F49620-79-C-0099

Jack Goldberg, Director
Computer Science Laboratory

David H. Brandin, Vice President and Director
Computer Science and Technology Division

Table of Contents



I INTRODUCTION

This is a final report covering progress on a 2-year research effort toward the development of new theorem-proving methods for program verification, and the empirical investigation of these methods in actual verification systems. In the last several years, interest in verification technology has been prompted by the tremendous cost of developing, debugging, and maintaining software. The creation of new software products is frequently characterized by time and cost overruns, and insufficient modifiability and reliability. Formal program verification offers a high payoff, though technically difficult approach to the solution of these problems. Admittedly, methods for proving the correctness of programs in a mathematical way have not yet been developed to the point of practicality for widespread everyday use. Nevertheless, much progress has been made in just the last 2 or 3 years, and the use of these techiques for verifying highly critical software now seems both practical and inevitable.

The experimental verification system we have developed under the present contract, in fact, is now successfully being used in the proof of correctness of the design of a sophisticated, fault- tolerant operating system developed under NASA support [Contract No. NAS1-15428]. We believe that the technology embodied in this experimental system is now nearly ready for transfer to a production environment staffed by well-trained (but not necessarily research-oriented) users. Although more research will be necessary to develop this system to the point of widespread use, we feel that the feasibility of verification as a practical technique is finally at hand, and we are currently seeking new Air Force support for the needed additional work.

The research conducted during the course of the project focused on methods for simplifying formulas of the kind that arise frequently in the verification of programs. The importance of simplification methods, as opposed to pure proof methods, was pointed up by verification work conducted under a previous AFOSR contract. Much of the effort in the latter years of that contract was directed toward developing fast, automatic deduction mechanisms in a system for verifiying JOCIT programs (RADC contract F30602-75-C-0042). Although the work on fast decision procedures enabled us to prove automatically many of the verification conditions and fragments of verification conditions generated in the RADC effort, it by no means facilitated automatic proof of all of the

formulas we encountered. The inadequacies were of two kinds: speed and generality. The first of these difficulties was made manifest by formulas whose Boolean structure produced a combinatorial explosion too large to be handled in a reasonable amount of time. The second deficiency was made apparent by large formulas that could be proven neither valid nor unsatisfiable by the decision procedures. For such formulas (usually verification conditions arising from improperly formulated inductive invariants), these procedures leave the user with no clue as to the reason why the given formula is not valid.

The work in developing simplification methods conducted under the current project addresses both of these difficulties. The algorithms embodied in the experimental system we implemented have been found to deal remarkably well with the propositional structure that typically arises in verification conditions. The method for simplifying interpreted formulas that was developed under the contract has been found quite effective in reducing the size of formulas whose validity could not be established, thus permitting the user to understand, through examination of the simplified formula, where the problem lies.

Research conducted under the project produced a substantial body of results in addition to those included in its original goals. Much of this additional work focuses on simplification methods based on canonical term rewrite systems investigated during the first year of the project. Additional work in the second year includes the investigation of deductive techniques for quantified formulas over the reals with inequalities.

The next few subsections describe the relation of this work to other Computer Science Laboratory work, and give an overview of results. Later sections, most of which are extracted from academic papers, form the main body of the report.

1.1. Relation to Other Computer Science Laboratory Projects

The Computer Science Laboratory at SRI has in the last several years conducted numerous projects involving program verification. The interaction among these various efforts has been of substantial mutual benefit. The current effort, for example, has benefited from the strong motivation for deduction tools provided by the more application- oriented projects. Conversely, our work in

the last 2 years has been, and continues to be, of utility in both our effort to prove the correctness of the SIFT fault-tolerant operating system, and in a project for the Rome Air Development Center to develop verifiers for several versions of the JOVIAL programming language. Other application-oriented projects have needed (and will need) sophisticated deductive tools for the verification of security properties of system software.

Our work for Rome Air Development center has been in progress almost continuously since 1975. Under contracts F30602-75-C-0042 and F30602-76-C-0204 ("Rugged Programming Environment", Phases RPE/1 and RPE/2), we developed early versions of program verifiers for a subset of JOVIAL/J3 and for JOCIT. A subsequent contract with RADC (F30602-78-C-0031) called for the development of a programming environment for JOVIAL-J73/I in which an Air Force programmer can design, implement, debug, and prove correctness for programs in this language. During the current reporting period, several aspects of the project work have been applied to the development of the Rugged Jovial Environment (RJE) program verification system. The RJE project is concerned with the application of program verification techniques to JOVIAL-J73 software.

Mutually beneficial relationships have arisen also with several other government-supported projects in this laboratory. Among these are:

- A Provably Secure Operating System (PSOS): The System, Its Applications, and Proofs. (SRI Project 4332, Contract DAAB03-75-C0399, for the U.S. Army. March 24, 1975 to February 11, 1977 plus subsequent work until August 1979).

- Kernelized Secure Operating System (KSOS)--Design and Verification. (SRI Project 6654, Contract MDA902-77-C-0333, Subcontract SC-606079-EW, for Ford Aerospace. August 3, 1977 to April 30, 1978).

- Formal Transformation of Computer Programs. (SRI Project 4079, Contract N00014-75-C-0816 for the Office of Naval Research. March 3, 1975 to May 31, 1980).

- Formal Methods for Fault Tolerance in Distributed Data Processing Systems. (SRI Project 7242, Contract DASG60-78-C-0046 for BMD ATC. February 27, 1978 to September 30, 1979).

- Investigation, Development, and Evaluation of Performance Proving for Fault-tolerant Computers. (SRI Project 7821, Contract NAS1-15528 for NASA-Langley. September 15, 1978 to September 15, 1981).

- Mechanizing the Mathematics of Computer Program Analysis. (SRI Project 8527, Grant MCS 79-04081 for the National Science Foundation. May 15, 1979 to May 15, 1982).

- Development of the Hierarchical Development Model (HDM). (SRI Project 1015, Contract N00039-79-C-0463 for the Department of the Navy. September 28, 1979 to September 30, 1980).

- OBJ-1, A Study in Executable Algebraic Formal Specification. (SRI Project 1350, Contract N00014-80-C-0296 for the Department of the Navy. August 18, 1980 to August 17, 1981).

- Hierarchical Methodologies for Communication Protocol. (SRI Project 1879, Contract NB80NAAE3398 for the National Bureau of Standards. August 21, 1980 to December 31, 1980).

- Towards an Editor and Interpreter for System Specifications. (SRI Project 2153, Letter dated 6-25-80 for Philips Research Laboratories. September 18, 1980 to September 1, 1981).

- PSOS Implementation Study — Consulting Report. (SRI Project 2958, Contract MDA904-81-C-0422 for U.S. Government. March 12, 1981 to September 15, 1982).

1.2 Overview of Results

The first year of the project was primarily concerned with Task 1 of the proposed work statement, i.e., the investigation of techniques for simplification of nonlogical expressions. Emphasis was placed on elaborating the method of interpreted implicants. The investigation was carried out in collaboration with Professor Donald Loveland, of Duke University. Preliminary results of this study were presented at the Fifth Conference on Automated Deduction held in July, 1980 at Les Arcs, France.

A substantial body of work on canonical term writing systems was also undertaken during the first year, under partial support of the project. Participating in this work were a number of visitors to SRI, including Gerard Huet and Jean-Marie Hullot (of INRIA, France), and Paul Gloess (SRI International Fellow). Four academic papers were produced, each touching on a different aspect of the use of rewrite systems to simplify formulas. Three of these papers ("Adding Dynamic Paramodulation to Rewrite Algorithms" (Gloess), "Canonical Forms and Unification" (Huet and Hullot), and "A Catalog of Canonical Term Writing Systems" (Hullot)) were presented at the Automated Deduction conference. "Equations of Rewrite Rules: A Survey" (Huet and Oppen) appeared in the proceedings of the 1980 Conference on the Foundations of

Computer Science held in Santa Barbara, Ca. Copies of these papers were included in the first year's report.

The second year of the project included work on all three tasks of the work statement. Further improvements to the method of interpreted implicants were devised. A complete description of the method, including these improvements, was issued as a Computer Science Laboratory technical report (CSL-117), and is included as Section II of this report. Another facet of the second year's work was the investigation of means for limited-expansion manipulation of propositional expressions. Several experimental computer programs were written in the Interlisp language and used to develop algorithms to minimize the combinatorial effect of case splitting in dealing with the propositional structure of formulas. The heuristics developed in this study were then incorporated within a full-blown experimental theorem prover, which has been used intensively in a number of verification efforts (listed in the previous subsection) in which the Computer Science Laboratory is now engaged. A description of this prover is given in Section III of this report, and the critical sections of the algorithms themselves, represented in Lisp, are supplied in an appendix. An extensive example illustrating the use of this system in the proof of the SIFT operating system has been provided by Michael Melliar-Smith and Richard L. Schwartz.

In addition to the work specifically called for by the project, the second year's activities included investigations in the related area of procedures for deciding formulas involving general equalities. A modified resolution procedure for this purpose was devised in collaboration with Prof. W. W. Bledsoe and Mr. Robern Neveln, both of the University of Texas. Section IV of this report describes the procedure in detail and gives completeness results.

II. Simplifying Interpreted Formulas[1]

D. W. Loveland[2] and R. E. Shostak[3]

Abstract

A method is presented for converting a decision procedure for unquantified formulas in an arbitrary first-order theory to a simplifier for such formulas. Given a quantifier-free disjunctive normal form (d.n.f.) formula, the method produces a simplest (according to a given criterion) d.n.f. equivalent from among all formulas with atoms in the original formula. The method is predicated on techniques for minimizing purely boolean expressions in the presence of "don't-care" conditions. The don't-cares are used to capture the semantics of the interpreted literals in the formula to be simplified.

Two procedures are described: a primitive version of the method that advances the fundamental idea, and a more refined version intended for practical use. Complexity issues are discussed, as is a nontrivial example illustrating the utility of the method. The last section describes an alternative to the first phase of the refined version that is preferable in certain cases.

---

[1] An abbreviated version of this paper was presented at the 5th Conference on Automated Deduction.

[2] Dept. of Comp. Sci., Duke University, Durham, NC 27706

[3] Computer Science Lab., SRI International, 333 Ravenswood, Menlo Park, CA 94025, (415) 326-6200 x2879; supported in part by AFOSR contract F49620-79-C-0099.

1. Introduction

The problem of simplifying logical expressions was first addressed in the early 1950s in the form of boolean minimization. The motivation at that time was to reduce as much as possible the number of components needed to realize a given switching circuit. Minimization techniques were developed to operate according to a variety of criteria, including the fewest literals in a sum-of-products or product-of-sums expression, the fewest terms, or the fewest terms and occurrences of literals.

The problem of simplifying logical expressions has resurfaced in the last few years in connection with program verification, synthesis, and allied concerns in artificial intelligence. In these applications, the expressions to be simplified are no longer merely propositional; they may contain interpreted predicates or function symbols. Even the problem of defining useful simplicity criteria for such formulas can be tricky, since the usual syntactic measures are sometimes misleading. For example, the formula $y \geq x \vee 5y \leq x+10$ (where x and y are understood to range over positive integers) is much more concise than the equivalent $(x=1 \wedge y=1) \vee (x=1 \wedge y=2) \vee (x=2 \wedge y=2)$, even though the latter is likely to be more useful in many theorem-proving situations.

Ideally, one would like a general-purpose method for simplifying formulas in arbitrary nonlogical theories with respect to arbitrary simplification measures. Though such a method is clearly too much to hope for, the approach described herein is a step in the direction of this goal. Our method may be viewed as a practical way of converting a decision procedure for unquantified

formulas in an arbitrary first-order theory to a simplifier for such formulas. Given a quantifier-free formula in d.n.f., it produces a simplest (according to any given reasonable criterion) d.n.f. equivalent from among all formulas whose atoms occur in the original formula. By "reasonable" criterion, we mean one according to which the deletion of a literal from a term or of a term from a disjunction always produces a simpler formula.

Before describing the approach, we might point out that simplification can often be accomplished merely by eliminating unsatisfiable disjuncts in a disjunctive normal form. (Note, in particular, that this technique necessarily reduces all unsatisfiable formulas to "false.") The elimination of such disjuncts is not, however, sufficient to produce a simplest form for nonvalid formulas. The difficulty is illustrated by the following formula from the theory of Presburger arithmetic with function symbols:

$$F \equiv (y \neq z) \vee (x \leq y \wedge x+y \leq 0) \vee (x \leq 1 \wedge f(z) \neq f(y)+1) \quad .$$

While none of the disjuncts of F is unsatisfiable, F does have a much simpler equivalent, namely

$$y \neq z \vee x \leq 1 \quad .$$

Isolated consideration of the terms in the d.n.f. expression is thus insufficient.

Our method is presented in five parts. Section 2 describes and proves the correctness of the standard procedure, a primitive version that advances the fundamental idea. A much more efficient version, called the modified procedure, is given and justified in Section 3. Section 4 gives a brief analysis of the computational complexity of the two versions, and Section 5 summarizes a nontrivial example that illustrates the utility of the modified method. The last section presents an alternative to the first phase of the

modified procedure that is beneficial in certain cases.

2. The Standard Procedure

The procedure given in this section takes as input a quantifier-free d.n.f. (c.n.f.) formula in a first-order theory and returns an equivalent d.n.f. (c.n.f.) expression with the property that no other such expression with atoms from the original formula is simpler with respect to a given reasonable (in the sense given earlier) measure of simplicity. The procedure works with any first-order theory for which the satisfiability of quantifier-free conjunctions of literals can be tested.

One can view the method as a nonlogical counterpart of the systematic minimization techniques developed for purely proposition formulas. In fact, the technique makes use of the method of prime implicants first described by Quine and McCluskey [3,4].

Our treatment assumes that a d.n.f. expression is to be found. One can obtain c.n.f. expressions using a dual method.

We begin with a brief review of Quine's method of prime implicants for purely propositional expressions. A more detailed account is given in [1].

Defn. A term is a conjunction of literals.

Defn. A term t_1 subsumes a term t_2 if each literal of t_2 is also a literal of t_1.

Defn. An implicant of a formula F is a term that implies F.

Defn. A prime implicant of a formula F is a term that implies F and subsumes no shorter term that implies F.

The fundamental interest of prime implicants is that any simplest d.n.f.

equivalent G for a propositional formula F must be a disjunction of prime implicants of F. To see this, suppose that some term t of G is not a prime implicant of F. Because t implies F but is not a prime implicant, t must subsume a shorter term t' that also implies F. The expression obtained from G by replacing t with t' is still equivalent to F, contradicting the assumption that G is simplest.

Several methods can be used to determine the set of prime implicants of a formula F. One such, called the method of iterated consensus [5,6] begins with the set of terms in a d.n.f. form of F. The nontautological resolvents of terms in the set are repeatedly formed and added to the set. At the same time, subsuming terms are deleted. When no new terms can be added that do not subsume existing terms, the set of prime implicants has been obtained.

Consider, for example, the formula F given by

$$F \equiv \bar{p}rs \vee \bar{p}q\bar{r}\bar{s} \vee pqrs \quad .$$

Resolving $\bar{p}rs$ and $\bar{p}q\bar{r}\bar{s}$ gives rise to $\bar{p}q\bar{r}$. Because $\bar{p}q\bar{r}\bar{s}$ subsumes $\bar{p}q\bar{r}$, the former can be deleted. Next, by resolving $\bar{p}rs$ with pqrs, one obtains qrs; pqrs can thus be deleted. Because no more terms can be added or deleted, the remaining terms, $\bar{p}rs$, $\bar{p}q\bar{r}$, and qrs, are the prime implicants of F.

Once the prime implicants of a formula have been found, a simplest d.n.f. expression can be obtained by determining a simplest subset of prime implicants whose disjunction is implied by the formula. Note that simplest disjunctions need not be unique; frequently several different combinations of prime implicants give rise to simplest equivalents. To discover these combinations, it is useful to classify the prime implicants into three

catagories:

- Core implicants are those that must appear in any such combination. If a given implicant does not imply the disjunction of all other implicants, it must be a member of the core.

- Absolutely eliminable implicants are those that imply the disjunction of the core implicants, and so can be ignored.

- Eliminable implicants are those that are neither core nor absolutely eliminable.

The various simplest equivalents differ only in their selection of eliminable implicants.

The most straightforward method of finding these equivalents involves constructing a table T whose rows are labeled by prime implicants and whose columns are labeled by the terms in the perfectly developed d.n.f. (In the perfectly developed d.n.f., each letter atom occurs (either signed or unsigned) in each term of the formula to be simplified.) A '1' is placed at T(t,u) if the prime implicant t is subsumed by term u, and a '0' otherwise. The core implicants are easily identified as those subsumed by at least one term that subsumes no other implicant; absolutely eliminable implicants are those subsumed only by terms that subsume at least one core implicant. All rows labeled by core and absolutely eliminable implicants are then canceled (deleted from the table), as well as all columns labeled by terms that subsume core implicants. The subsets of remaining implicants sufficient to cover the remaining columns are then enumerated exhaustively and a simplest one is selected.

Our procedure for simplifying interpreted expressions depends on an

elaboration of the method just described that can handle so-called "don't-care" conditions. In the application of minimization techniques to digital design it is sometimes useful to exploit situations in which certain assignments to the variables of an expression to be simplified are not actually realized. For such assignments, the value of the simplified expression can be arbitrary. As one might expect, greater simplification can often be obtained if one relaxes the requirement that the simplified expression be equivalent to the original, so as to necessitate equivalence only for assignments other than the don't-cares.

The treatment of don't-care conditions requires two slight modifications of the basic method. First, for purposes of generating prime implicants, the d.n.f. form of the formula to be simplified is augmented by disjoining to it a term for each don't-care condition. If, for example, p=T, q=F, r=T is a don't-care input, the term $p\bar{q}r$ is added. Second, the terms in the perfectly developed d.n.f. that imply don't-care conditions are omitted from the prime-implicant matrix.

Suppose it is wished, for example, to simplify the formula $F \equiv p \vee qr$ with respect to don't-care conditions {p=F, q=T, r=F} and {p=T, q=F, r=T}. We first find the prime implicants of the augmented formula $p \vee qr \vee \bar{p}q\bar{r} \vee p\bar{q}r$. Using the method of iterated consensus, $p\bar{q}r$ can be eliminated immediately because it is subsumes p. Resolving p against $\bar{p}q\bar{r}$, $q\bar{r}$ is obtained. Since $\bar{p}q\bar{r}$ subsumes $q\bar{r}$, $\bar{p}q\bar{r}$ can now be eliminated. Resolving $q\bar{r}$ against qr yields q, which permits the elimination of both $q\bar{r}$ and qr. We are therefore left with the prime implicants p and q. The prime implicant table will contain rows for p and q and columns for all the terms in the

perfectly-developed d.n.f. for F (namely, pqr, $p\bar{q}r$, $\bar{p}qr$, $\bar{p}\bar{q}r$, $\bar{p}\bar{q}\bar{r}$) other than the don't-care term $\bar{p}\bar{q}r$. It is easy to verify that both p and q are core implicants (q is subsumed by $\bar{p}qr$ and p by the remaining terms), hence the simplified form is just $p \vee q$.

Our application of this method to the problem of simplifying interpreted expressions is predicated on the use of don't-care conditions to encode the semantics of the terms appearing in the expressions. The basic idea is to treat the interpreted formula to be simplified as if it were purely propositional (i.e., as if interpreted terms were actually uninterpreted), except that all unsatisfiable (with respect to the interpreted semantics) conjunctions of literals with atoms occurring in the formula are treated as don't-cares.

The procedure is easily understood in the context of a small example. Suppose, then, that the formula F to be simplified is just

$$x \leq y \vee (z>0 \wedge x+2z-y \geq 3) \quad ,$$

where all variables range over nonnegative integers.

If we let p, q, r denote the atoms $x \leq y$, $z>0$, and $x+2z-y \geq 3$, respectively, F can be written $p \vee qr$.

Now consider the eight possible assignments of truth values to p, q, r: pqr, $pq\bar{r}$, $p\bar{q}r$,..., $\bar{p}\bar{q}\bar{r}$. If each term were submitted to a refutation procedure for quantifier-free Presburger arithmetic, it would be found that all assignments other than $\bar{p}\bar{q}\bar{r}$ and $\bar{p}qr$ are satisfiable. The question of simplifying F thus becomes that of finding the simplest propositional equivalent of $p \vee qr$ subject to the don't-care conditions $\bar{p}\bar{q}\bar{r}$ and $\bar{p}qr$. Having solved this problem

in the propositional example above, we may conclude that p $\vee$ q, i.e., $x \leq y \vee z > 0$, is a simplest equivalent. (Note, incidentally, that since p and q are core implicants, the fact that p $\vee$ q is simplest does not depend on the simplicity measure.)

The standard method may be summarized as follows:

1. Let A be the set of atoms occurring in the formula F to be simplified, and let T be the set of terms representing the $2^{|A|}$ truth assignments to A. Using a refutation procedure for the theory in question, determine the unsatisfiable subset U of T.
2. Using the method of prime implicants, find a simplest (with respect to the desired reasonable measure) formula that is truth-functionally equivalent to F modulo the don't-care set U.

Our proof that the standard method does indeed produce a simplest semantic equivalent for F among all formulas with atoms in A requires a few definitions.

In the following, we will assume that F and F' are both quantifier-free formulas in a first-order theory Th, that as before, A is the set of atoms occurring in F, and that the atoms of F' are contained in A.

Defn. If S is a set of truth assignments to A, we say that F and F' are truth-functionally equivalent with respect to S if $F \equiv F'$ evaluates to true for each truth assignment in S.

Defn. The full term of a truth assignment m to A is a conjunction of literals, one for each atom in A, such that each atom true in m occurs positively, and each atom that is false in m occurs negatively.

Defn. A truth-assignment to A is semantically consistent if the corresponding full term is satisfiable in Th.

Claim. F and F', are equivalent in Th iff they are truth-functionally equivalent with respect to the set of semantically consistent

truth assignments to A.

Pf. => Suppose F and F' are equivalent in Th. Let m be any semantically consistent truth assignment. Since m is semantically consistent, its corresponding full term is true in some model I of Th. Since each literal of F ≡ F' is assigned the same value by I as it is by m, F ≡ F' must have the same value in I as it does in m. Since F ≡ F' is valid in Th, it is true in I, hence in m.

<= Suppose F and F' are not equivalent in Th. Then F ≡ F' must be false in some model I of Th. Let m be the truth assignment that gives each atom of A the value given it by I. I satisfies the full term corresponding to m, so m is semantically consistent. Since m gives each atom of A the same value as I, F ≡ F' is false in m, hence F and F' are not truth-functionally equivalent with respect to the set of consistent truth assignments of A.

Q.E.D.

It follows as a corollary the claim that among all quantifier-free formulas of Th with atoms in A, a simplest equivalent to F, according to any measure, must be a simplest truth-functional equivalent to F with respect to the set of semantically consistent truth assignments to A. The correctness of the standard procedure follows immediately, once it is observed that (i) the don't-care procedure finds a simplest truth-functional equivalent with respect to the complement (in the space of all assignments to A) of the given don't-care set, and (ii) the complement of the don't-care set, in the standard procedure, is the set of semantically consistent truth assignments.

3. The Modified Procedure

Because the problem solved embeds the satisfiability question for propositional formulas, any version of the procedure requires (at least)

exponential time in input formula length in the worst case (based on present-day knowledge). This section details refinements of the standard procedure, however, that improve performance greatly in many situations. The standard procedure may nevertheless be preferable when there is a substantial number of multiple occurrences of atoms of F.

Our measure of effort will be taken as the number of calls to the refutation procedure. That this is the best measure is arguable since some refutation procedures can be so quick as to have the boolean manipulation dominate the cost. However, our methods are independent of the refutation procedure used and most such procedures require a significant interval of time per call (which may be only a second, but is nevertheless significant when hundreds of calls are made). Moreover, except for the iterated consensus (resolution) section, total effort is proportional to the number of calls.

The greatest potential for performance gain follows from the requirement of the standard procedure that all conjunctions to be processed must be evaluated by the refutation procedure before serious boolean processing begins. Although we improve the "worst-case" situation somewhat (worst-case with respect to the various chances that simplification may occur), we greatly improve the cost of processing a typical formula, especially when no simplification does occur. We are left at least with the situation that high cost is associated with definite gain.

For purposes of explanation, it is convenient to consider the standard procedure as consisting of two phases: in Phase 1, the unsatisfiable truth assignments are determined and the prime implicants of the formula augmented

by don't-care terms are generated; in Phase 2, the prime-implicant table is created and a simplest set of implicants implied by the original formula is chosen. The improved procedure refines both of these phases.

The main improvement to Phase 1 turns upon the observation that it is unnecessary to test all truth assignments for satisfiability. In particular, the assignments that subsume terms of the original formula need not be tested, since these assignments would be discarded in the iterated consensus procedure anyway. In our earlier example, for instance, five of the eight assignments (namely pqr, $pq\bar{r}$, $p\bar{q}r$, $p\bar{q}\bar{r}$, and $\bar{p}qr$) subsume either p or qr, leaving only three ($\bar{p}\bar{q}\bar{r}$, $\bar{p}q\bar{r}$, $\bar{p}\bar{q}r$) to be submitted to the refutation procedure.

Described in Section 5 is another refinement of Phase 1 that further lowers the required number of calls to the procedure, but at the cost of possibly missing significant simplifications.

The improved Phase 2 procedure is equivalent to the standard one, but is substantially more efficient in most cases. It appears not to have been considered for boolean minimization because "don't-care" conditions are traditionally given rather than computed.

The procedure is defined using an auxiliary predicate P(X,Y), where X and Y are sets of terms. Letting $Y=\{t_1,t_2 \ldots t_k\}$, P(X,Y) is computed by enumerating all terms of the form

$$C \wedge L_1 \wedge L_2 \ldots \wedge L_k \quad ,$$

where C is the conjunction of all terms in X and each L_i is the complement of some literal in t_i. The enumerated terms are tested one by one for satisfiability. P returns "true" if one is found to be satisfiable, and

returns "false" otherwise. The key property of P is that P(X,Y)=false iff $C \supset t_1 \vee t_2 \vee \ldots \vee t_k$.

If for example, X={a,bc} and Y={cde,gh}, the terms $abc\overline{cg}$, $abc\overline{ch}$, $abc\overline{dg}$, $abc\overline{dh}$, $abc\overline{eg}$, $abc\overline{eh}$ are enumerated. Note that the first two of these are syntactically unsatisfiable, and so do not require calls to the refutation procedure. If it were found, for instance, that $abc\overline{dg}$ is satisfiable, the evaluation could terminate after this one call, returning "true."

The improved Phase 2 procedure is as follows. Let I be the set of prime implicants computed by Phase 1, and let I' be obtained by deleting from I all of its unsatisfiable members. (Computing I' from I thus requires applying the refutation procedure to each member.) A modified prime-implicant table T_m is now constructed whose rows are labeled with members of I' and whose columns are labeled by sets of terms. The columns are created dynamically in the following way:

1. Initialize the table by creating a column for each term in I', with the singleton set of that term as label.

2. Fill in each new column as follows. If P(X,I'-X) evaluates to false, where X is the set labeling the column to be filled in, enter '*' in each row position (indicating a cancelled column). Otherwise, for the row labeled by implicant u, enter '1' if $u \in X$ and '0' if $u \notin X$.

3. For each two cancelled columns with labels X_1, X_2, create a new column, if one does not already exist, labeled by $X_1 \cup X_2$.

4. Repeat Steps (2) and (3) until no new columns can be added.

5. Select prime implicants to define a simplest equivalent to F as in the standard procedure--i.e., choose a simplest set S of prime implicants such that for every uncancelled column X, there exists

an $s \in S$ such that $T_m(s,X)=1$.

We illustrate the modified procedure with the earlier example:

$$F = a \vee bc \vee de$$

where

a: $y \neq z$
b: $x \leq y$
c: $x+y \leq 0$
d: $x \leq 1$
e: $f\bar{z} \neq fy+1$.

Phase 1.

The truth assignments not subsuming terms in F are $\bar{a}\bar{b}\bar{c}\bar{d}\bar{e}$, $\bar{a}\bar{b}\bar{c}\bar{d}e$, $\bar{a}\bar{b}c\bar{d}\bar{e}$, $\bar{a}\bar{b}cde$, $\bar{a}b\bar{c}\bar{d}\bar{e}$, $\bar{a}b\bar{c}\bar{d}e$, $\bar{a}bc\bar{d}\bar{e}$, $\bar{a}bcde$, and $abcde$. Of these nine, all but $\bar{a}\bar{b}\bar{c}\bar{d}\bar{e}$, $\bar{a}b\bar{c}\bar{d}\bar{e}$, and $\bar{a}bcde$ are found by a refutation procedure to be unsatisfiable. The iterated-consensus process is applied to F augmented by the six "don't-cares" to obtain the set $I = \{a, bc, d, \bar{e}\}$ of prime implicants.

Phase 2.

Each member of I is tested and found satisfiable, so $I' = I$. The modified table T_m is initialized with rows and columns labeled by members of I'. Steps (2) and (3) of the Phase 2 procedure are now applied repeatedly to form the table shown below.

	{a}	{bc}	{d}	{$\bar{e}$}	{bc, $\bar{e}$}
a	1	*	0	*	*
bc	0	*	0	*	*
d	0	*	1	*	*
$\bar{e}$	0	*	0	*	*

Justification for the table is summarized below:

1. Initialize, creating columns labeled {a}, {bc}, {d}, {$\bar{e}$}
2. Fill in columns:

Column{a}:
P({a}, {bc, d, $\bar{e}$}:
conjunction tested: $a\bar{b}\bar{c}de$ satisfiable
= true
Fill in standard way

Column{bc}
P({bc}, {a, d, $\bar{e}$})
conjunction tested: $bc\bar{a}de$ unsatisfiable
= false
cancel column

Column{d}:
P({d}, {a, bc, $\bar{e}$}):
conjunction tested: $\overline{dabe}$ satisfiable
= true
Fill in standard way

Column {$\bar{e}$}:
P({$\bar{e}$}, {a, bc, d})
conjunctions tested: $\overline{eabd}$ unsatisfiable
$\overline{eacd}$ unsatisfiable
= false
cancel column

3. Create new column labeled {bc,$\bar{e}$}

2. (Repeated). Fill in new columns:

Column {bc, $\bar{e}$}
P({bc, $\bar{e}$}, {a, d})
conjunction tested: $\overline{bcead}$ unsatisfiable
= false
cancel column

3. (Repeated). No new columns

5. Core implicants a,d cover all uncanceled columns.

The simplified form is thus a $\vee$ d, i.e., F = y$\neq$z $\vee$ x=1.

(Note that here only 19 calls to the refutation procedure were required, as against 32 for the standard procedure.)

The correctness of the modified phase 2 procedure is established by the following theorem.

Theorem The standard and modified procedures yield the same minimal formulae.

Proof: Because don't-care terms cannot label columns of the table created in the standard procedure, any row of that table headed by an unsatisfiable implicant must contain only zeroes, and so cannot participate in an implicant selection. Letting T_s denote the table obtained by omitting such rows, it thus suffices to show that T_s and T_m (the table generated in the modified procedure) yield the same implicant selections.

Note that the rows of both T_s and T_m are labeled with the members of the satisfiable subset I' of implicants generated in phase 1. We will assume without loss of generality that these implicants are assigned to rows in the same order for the two tables.

Let T_s^* be the table obtained from T_s by removing any column v for which there is another column v' with fewer 1's and such that v has a 1 in every row position that v' does. We claim that it is enough to show that $V(T_s^*) \subseteq V(T_m)$, and $V(T_m) \subseteq V(T_s)$, where V(T) denotes the set of uncanceled column vectors of table T. To see this, note that a prime implicant selection need only meet the condition that every uncancelled column vector have a 1 in some row labeled by a selected implicant. Any implicant selection that satisfies this condition for T_m must, from $V(T_s^*) \subseteq V(T_m)$, satisfy it for T_s^*, and hence for T_s. Conversely, any selection that satisfies the condition for T_s must, from $V(T_m) \subseteq V(T_s)$, satisfy it for T_m.

To show that $V(T_m) \subseteq V(T_s)$, let v be an arbitrary column vector in $V(T_m)$ and suppose v occurs in T_m with label X. Let t be the conjunction of terms in X. Since P(X,I'-X) is true, there exists a satisfiable conjunction C subsuming t with the complement of (at least) one literal from each term in I'-X. Let $A_1,\ldots,A_k$ be the atoms of F missing from C. The conjunction $C \wedge (A_1 \vee \overline{A}_1) \wedge (A_2 \vee \overline{A}_2) \wedge \ldots \wedge (A_k \vee \overline{A}_k)$ is satisfiable, so at least one term u in its disjunctive expansion must be satisfiable. Because u is a full term that

subsumes a conjunction of implicants of F, u must occur in the perfectly-developed d.n.f. of F. The vector labeled by u in T_s, moreover, must be v, giving $v \in V(T_s)$ as required.

For $V(T_s^*) \subseteq V(T_m)$, let v be a column vector in $V(T_s^*)$, and suppose v occurs in T_s^* with label t. Let X be the set of implicants in I' that are subsumed by t. Because t is a full term, each implicant of I' not in X must have a literal whose complement occurs in t. Let C be the conjunction of all literals in X, and for each implicant s not in X, at least one literal of t whose complement occurs in s. C must be satisfiable because t is. Since C is a conjunction tested by P(X, I' - X), P(X, I' - X) must therefore be true. Thus if X labels a column in T_m, that column must be uncancelled, hence v a column vector.

It suffices to show, then, that X does indeed label a column of T_m. So suppose not. Then there exists a proper subset Y of X that labels an uncancelled column of T_m with 1's in only some of the rows in which v has 1's. Since $V(T_m) \subseteq V(T_s)$, this vector also occurs in T_s. But then v could not occur in T_s^*, giving a contradiction.

Q.E.D.

4. Complexity Issues

While it is difficult to obtain quantitative measures of the improvement afforded by the modified procedure, some calculations can be made under certain simplifying assumptions. Our analysis will consider that the formula F to be simplified has n terms, each with m literals, and that no atoms in F have multiple occurrences.

For the standard procedure, exactly 2^{mn} calls to the refutation procedure are made in Phase 1 and, of course, none are made in Phase 2.

For the modified procedure, calls are made in both phases. In Phase 1, a call is made for each truth assignment (to the mn atoms of F) that does not subsume a term of F. Each truth assignment may be viewed as a choice. for each term, of one of 2^m assignments to the atoms of that term. Because all but one of these 2^m assignments are permissible, a total of $(2^m-1)^n$ calls is made in Phase 1.

The number of calls made in Phase 2 depends on the set I of prime implicants discovered in the first phase. To obtain a rough idea of Phase 2 behavior, let us assume I contains p implicants, each with q literals, and that $p \leq n$, $q \leq m$. (We have found this assumption to be valid in practice.)

Phase 2 first requires that each of the p implicants be tested for satisfiability. The remainder of Phase 2 may require zero calls (if all prime implicants are unsatisfiable). Assuming that p prime implicants are satisfiable, we may need as few as p more calls (if all tested conjunctions are satisfiable) or as many as $(q+1)^p-1$ more calls (if all tested conjunctions are unsatisfiable). The lower bound holds because for each singleton set X, P(X,I'-X) will return "true" after one call and Step 3 provides no new columns beyond the p initial columns. Thus, a total of p calls is made. The upper bound holds because each conjunction tested contains for each of the p prime implicants either the prime implicant itself or the complement of one of the q literals of the implicant. In the one unrealizable case, no prime implicant occurs in the conjunction. Using $p \leq n$, $q \leq m$, we have a worst-case bound of $(m+1)^n-1$, and a best-case bound of 2n.

It is worth noting that the total worst-case cost for the modified procedure

is almost always less than that for the standard procedure ($(2^m-1)^n+n+(m+1)^n-1$ versus 2^{mn}) for reasonable m and n. However, the primary value of the modified procedure is that often m is small enough (typically averaging about 1.5) so that Phase 1 cost is moderate. Moreover, a general mix of candidate formulas includes many that are not simplifiable and with the cost of Phase 2 close to 2n.

5. An Example

This section gives a summary of a less trivial example. The example illustrates that quite striking reductions can be obtained in innocent-looking formulas.

Consider

$$F \equiv y>\max(2,z) \vee y>1+z \vee (y\neq 0 \wedge y\leq -1) \vee (y\neq 0 \wedge y\neq z) \vee y=0 \vee (z\neq 1 \wedge y\neq 1) \quad .$$

Phase 1: Use of modified procedure requires 3 calls, and results in prime-implicant set:

$$\{y>\max(2,z),\ y>1+z,\ y=0,\ y\leq -1,\ y\neq z,\ z\neq 1,\ y\neq 1\} \quad .$$

Phase 2: Modified procedure requires 63 calls.

Result: $F \equiv z\neq 1 \vee y\neq 1$.

The standard procedure requires 128 calls.

To balance this example, we consider two formulas with similar structure to F, but where little simplification occurs. The letters A, B, ... represent semantically unrelated atoms.

$$F_1 \equiv A \vee B \vee (\bar{C} \wedge D) \vee (\bar{C} \wedge E) \vee C \vee (G \wedge H)$$

(which simplifies to $F_1 \equiv A \vee B \vee D \vee E \vee C \vee (G \wedge H)$) ,

$$F_2 \equiv A \vee B \vee (C \wedge)D \vee (W \wedge E) \vee Z \vee (G \wedge H) \quad .$$

F_1 produces 3 Phase 1 calls and 12 Phase 2 calls. F_2 produces 27 phase 1 calls and 12 Phase 2 calls. The standard procedure requires 128 and 512 calls, respectively.

6. Phase 1 Alternative

We conclude with a description of an alternative phase 1 procedure. The need for improvement relative to the procedures described earlier is strong when there are numerous multiliteral terms. Although the worst-case cost is little improved, we again are able to reduce costs when few conjunctions of literals of the given formula F are unsatisfiable.

The reduction is obtained at the tradeoff of the guarantee of finding all prime implicants -- the alternative procedure detects only prime implicants that are subterms of terms of F. This tradeoff is more favorable than it might at first seem, since proper subterm implicants have the advantage of guaranteeing simplification. Moreover, nonsubterm conjunctions of literals with atoms in F are more rarely prime implicants, and are especially less likely to appear in the final simplified formula. A nonsubterm implicant must be implied by some other implicant in order to appear in the final simplification. This rather strong constraint is automatically satisfied by subterm implicants.

The alternative procedure is carried out in two stages. First, iterated consensus is applied to F as before, but without first computing and adding in don't-care terms. Terms in the resulting set of implicants that are not

subterms of terms in F are discarded. (Alternatively, but not necessarily equivalently, one could modify iterated consensus to disallow resolvents other than subterms of terms in F. This would tend to reduce the cost of the first stage at the expense of the second stage, and might be preferable in certain instances.)

In the second stage, each subterm implicant is tested for primeness. An implicant t is tested by determining, in a manner described momentarily, whether it has a subterm (i.e., a subterm with one fewer literal) that is also an implicant. If not, t is prime, and so is included in the output of phase 1. Otherwise, t is discarded in favor of its subterm implicants, which are themselves tested for primeness. Proceeding depth-first, one has the option of discontinuing subterm checking if a desirable subterm implicant (such as a unit) is determined.

The key aspect of the alternative procedure is the use of the P predicate described earlier to determine quickly whether a given subterm of a subterm implicant t is also an implicant. Letting $l_1, l_2 \ldots l_k$ denote the literals of t, and $l_1, l_2 \ldots l_{k-1}$ the literals of the subterm in question, the determination is made by computing $P(\{l_1, l_2, \ldots l_{k-1}, \overline{l}_k\}, \hat{F} - \{u\})$, where $\hat{F}$ is the set of terms of F and u is the term of F (or one of possibly several) of which t is a subterm. As we will show P computes to false if and only if the subterm is an implicant.

For illustration, consider the earlier example formula $F = a \vee bc \vee de$, where

a: $y \neq z$
b: $x \leq y$
c: $x + y \leq 0$
d: $x \leq 1$
e: $fz \neq fy + 1$

In this example, the iterated consensus stage has no effect, leaving the terms of F as the set of implicants to be tested. The unit literal a has no subterms and so is prime. It remains to test bc and de:

$P(b\bar{c}, \{a,de\})$:

conjunction tested: $\overline{bcad}$ satisfiable

= true

$P(\bar{b}c, \{a, de\})$:

conjunction tested: $\overline{\bar{b}cad}$ satisfiable

= true

$\therefore$ bc is prime

$P(d\bar{e}, \{a, bc\})$:

conjunctions tested: $\overline{deab}$ unsatisfiable

$\overline{deac}$ unsatisfiable

= false

$P(\bar{d}e, \{a, bc\})$:

conjunction tested: $\overline{\bar{d}eab}$ satisfiable

= true

$\therefore$ d is a prime implicant, de and e are not.

We have, then, that a, bc, and d are prime implicants. The implicant $\bar{e}$ is not found; however, $\bar{e}$ does not appear in the final simplified formula, which is $a \vee d$. Note that five calls to the refutation procedure are made, as compared with nine calls by the modified procedure.

The use of the P predicate is justified in the following lemma.

Lemma Suppose $u = l_1 l_2 \ldots l_k$ is a term of F and $t = l_1 l_2 \ldots l_r$, $2 \leq r \leq k$, is an implicant of F. Then $l_1 \wedge \ldots l_{r-1}$ is also an implicant of F iff $P(\{l_1 l_2 \ldots l_{r-1}, \bar{l}_r\}, \hat{F} - \{u\}) = \text{false}$.

Pf. => If $P(\{l_1 l_2 \ldots l_{r-1}, \bar{l}_r\}, \hat{F} - \{u\})$ is true, then there is a satisfiable conjunction $l_1 \ldots l_{r-1} \bar{l}_r \bar{l}_{u(1)} \ldots \bar{l}_{u(i)} \ldots l_{u(n)}$, where $l_{u(i)}$ is a literal in term u(i) of F, with term u omitted from the indexing. But since l_r is a literal of u, every term of F is falsified so $l_1 \wedge l_2 \wedge \ldots \wedge l_{r-1} \not\supset F$ does not hold, contrary to assumption.

<= If $l_1 \wedge \ldots \wedge l_{r-1}$ is not an implicant, there is an interpretation of F verifying $l_1, \ldots, l_{r-1}$ but falsifying at least one literal $l_{u(i)}$ of each term u(i) of F. But since t is an implicant, so that $l_1 \wedge \ldots \wedge l_r \supset F$, l_r must be falsified in this interpretation. But then $l_1 \ldots l_r \bar{l}_r \bar{l}_{t(1)} \ldots \bar{l}_{t(n)}$ is satisfiable, contradicting $P(\{l_1, l_2 \ldots l_{r-1}, \bar{l}_r\}, \hat{F} - \{u\}) =$ false.

Q.E.D.

To obtain some general measure of the improvement afforded by the alternative phase 1, we again count calls to the refutation decision procedure, and consider formulas with n terms of m literals each, with every atom having a unique occurrence in F.

When all conjunctions are satisfiable only one conjunction is tested for each of the m subterms for a cost of nm. If all conjunctions are unsatisfiable up to 2^m-2 subterms can be tested for each term, each checking m^{n-1} conjunctions for a total of $n(2^m-2)m^{n-1}$ calls, (although in this case a depth-first search would hold the cost to m^n.) A more useful observation is that finding one new subterm prime implicant costs m^{n-1} calls.

We emphasize again that while gains over the modified phase 1 method can be appreciable when a number of multiliteral terms exist and little simplification occurs, this must be weighed against the possibility of missed prime implicants of value. The alternative procedure also has less value when

many multiple occurrences of literals are found in the given formula. To expedite this case, each conjunction should be checked for complementary literals before submission to the decision procedure.

It should be clear that the procedure we have described is but one of a number of alternatives. For large formulas one may check only small subterms (using $P(t_1, \hat{F})$ rather than $P(t_1, \hat{F} \cdot \{u\})$, where t_1 is a subterm of t, when necessary). If one wishes to consider all subterms with complementation of literals introduced (so that $\bar{e}$ would be found as a prime implicant in our example) then testing should be on conjunctions t_1 each of which contains all literals of the term t or their compliments. Resolution is then employed on the conjunctions seen to be implicants. The worst-case cost of $n(2^m-1)m^{n-1}$ is only slightly worse than for subterm testing alone, but often all 2^m-1 patterns need be tested. (However, many $P(t,\hat{F} - \{t\})$ may test as few as one conjunction.)

Truly low-cost maximal simplification using refutation decision procedures is unlikely. However, we believe this paper shows that, given the speed of the best existing refutation procedures, simplification of expressions that occur in practice is currently feasible.

7. REFERENCES

1. Bartee, T. C., Lebow, J. L., Reed, I. S., Theory and Design of Digital Machines, McGraw-Hill, New York (1962).

2. McCluskey, E. J., "Minimization of Boolean Functions," Bell System Tech. Journal, Vol. 35, pp. 1417-1444 (Nov. 1956).

3. Quine, W. V., "The Problem of Simplifying Truth Functions," Am. Math. Monthly, Vol. 59, pp. 521-531 (Oct. 1952).

4. Quine, W. V., "On Cores and Prime Implicants of Truth Functions," Am. Math. Monthly, Vol. 66, pp. 755-760 (Nov. 1959).

5. Samson, E. W. and Mills, B. E., "Circuit Minimization: Algebra and Algorithm for New Boolean Canonical Expressions," AFCRC-TR-56-110, Cambridge, MAssachusetts (1954).

III AN EXPERIMENTAL PROVER

In the second year, much effort was devoted to the development of an experimental theorem prover with the purpose of testing and refining the theoretical results of the project in a practical setting. The resulting verification system has been used and continues to be used extensively in a NASA-supported effort to verify the correctness of a complex fault- tolerant operating system. Participants in this effort include D. Hare, Dr. K. Levitt, P. M. Melliar-Smith, and Dr. R. Schwartz, all of whom have been instrumental in the development of the prover. The use of the system for this effort has been so successful that we are currently seeking support for the further research and development needed to create a production version.

The system consists of a decision algorithm-based theorem-prover for typed predicate calculus, together with a set of environment support functions. Formulas in the typed theory are constructed from:

- Integer, real, rational and user-defined constants
- Integer, real, rational and user-defined variables
- The propositional connectives IMPLIES, NOT, AND, OR, IFF
- The first-order connectives FORALL, EXISTS
- The three-placed IF construction
- The relational operators EQUAL, LESSP, LESSEQP, GREATERP, GREATEREQP
- The arithmetic operators PLUS, TIMES, MINUS, DIFFERENCE
- Uninterpreted function symbols of INTEGER, RATIONAL, and user-defined types

The theory also includes a definitional facility that permits user-created conservative extensions.

One of the more interesting (and powerful) aspects of the theory over which the prover operates is the provision for user-defined types. This facility permits the abstract data type information associated with a program that is to be verified to be carried down to the level of the verification conditions. This information is passed to the theorem prover through explicit type declarations for variables and function symbols occuring in the formulas to be proved. The proof process includes a typechecking phase that verifies the syntactic correctness of the formula. Type information is extracted during

this phase, and incorporated into a TYPE MODULE that the theorem prover proper subsequently consults during the proof process.

It should be noted that while the language we have described is first-order (i.e., includes quantifiers), the decision procedures that underly the prover operated exclusively on ground (unquantified) formulas. The prover automatically skolemizes a quantified formula to obtain a ground formula, and relies on the user to provide the necessary instantiations of the quantified variables in the resulting Skolem form.

The prover has been found to be able to prove remarkably complex (with respect to syntactic measures) verification conditions on the order of several seconds. The fast response is due in large part to a considerable amount of experimentation with the mechanism used to process the propositional super-structure of the formula to be proved.

Perhaps the main lesson learned from this experimentation was that vast changes in speed performance could result from apparently minor "fine tuning" of this mechanism. Because the modifications to which performance was sensitive were often extremely slight, it is difficult to draw conclusions about how one should go about treating propositional structure in general. Nevertheless, a number of ideas were developed that are of general interest. First, it was determined that success in handling propositional structure depends on a delicate balance between simplification and proof. "Proof", here, refers to an attempt to reduce a formula or subformula to either "true" or "false"; failure of the attempt produces no other information. Simplification, on the other hand, may result in reducing a formula that can be proved neither true nor false to an equivalent formula that is at least syntactically more tractable. The utility of simplification as a subprocess of proof is well established; it proved to be especially so in our case, because it often obviated the case-splitting that is more often than not responsible for combinatorial explosion in the reduction of propositional structure. As an illustration, consider the following propositional expression E:

```
E = (AND P (OR (NOT P)(NOT Q))(OR Q (AND (NOT P)(NOT Q)) R)
          (OR (NOT P) Q (NOT R)))
```

We wish to reduce E to TRUE or FALSE. Ordinary case splitting, even when preceded by recursive reduction of subexpressions to TRUE or FALSE when

possible, produces 1 x 2 x 3 x 3 = 18 cases (conjuncts) in the disjunctive normal form. By recursively simplifying, however, E can be treated without any case-splitting at all. In particular, simplification of the disjunct (OR (NOT P)(NOT Q)) in the context of the unit literal P produces a second unit literal (NOT Q). Simplification of the next disjunct in the context of the two unit literals P and (NOT Q) produces a third unit literal R. Simplification of the last disjunct then produces a contradiction, thus reducing E to false.

Unfortunately, simplification is much more time consuming than proof, because, as illustrated in our example, the results of each simplification must be repeatedly applied to obtain other simplifications. We found that just the right balance had to be struck between simplification and proof in the internal structure of the propositional reduction mechanism to obtain the benefits of simplification without paying too dearly for the additional analysis it requires.

A second idea developed from our experimentation is the utility of the "FAST.PROVE" strategy. FAST.PROVE is a subalgorithm of our propositional manipulator that attempts to reduce a formula (or subformula) without permitting any case-splitting at all. Although FAST.PROVE is, of course, incomplete, it was found to be quite effective as a kind of preprocessor; a given formula would be subjected to FAST.PROVE at each level of its tree structure before any case splitting would be undertaken at all. Once again, it was discovered that a delicate balance had to be maintained in order not to waste too much time in the case where the FAST.PROVE component was not successful. As in the case of simplification, the criticality of this balance is due to the recursive structure of the prover as a whole, which greatly magnifies the effect, for better or worse, of any computation that is carried out at each level of the recursion.

The remainder of this section illustrates the operation of the theorem prover on some examples. The first series of examples involve simple mathematical identities, and are included to exemplify operation of the prover. The second series is extracted from the design proof of the SIFT operating system, and was kindly furnished by Melliar-Smith and Schwartz. A partial listing of the propositional simplifier portion of the prover is supplied in an appendix for the benefit of those interested in the details of its operation. In the

following, annotations in brackets are not part of the user-machine dialogue, but were inserted after the fact for the purposes of explanation. Lines headed by numbers show commands issued by the user.

```
2_DSV(NUMBER X)

3_DSV(NUMBER Y)

4_DSV(NUMBER Z)
```

[In the three DSV (Declare Symbol Variable) commands above, the user declares X, Y, and Z to be numbers (i.e., reals)]

```
5_DD(NUMBER MAX(X Y)(IF (LESSP X Y) Y X))
```

[This Declare Definition command defines the function MAX that takes two numbers as arguments and returns a number. Note that the IF construct that provides the definition defines MAX in the usual way.]

```
6_DF(MAX.COMMUTE (EQUAL (MAX X Y)(MAX Y X)))
```

[This Declare Formula command associates the name MAX.COMMUTE with the given formula. The system typechecks the formula, and would issue an error message if it were found to be ill-formed.]

```
7_PR(MAX.COMMUTE)
----------Proving----------
602 conses
.7 seconds
Proved
```

[The user now invokes the prover on the formula MAX.COMMUTE. After .7 CPU seconds, the prover returns Proved, and indicates the number LISP conses required by the proof.]

```
8_DF(MAX.ASSOC (EQUAL (MAX X (MAX Y Z))(MAX (MAX X Y) Z)))
9_PR(MAX.ASSOC)
----------Proving----------

32343 conses
26.05 seconds
Proved
```

[The IF structure in the definition of MAX produces a great deal of propositional case-splitting in the proof of this formula, accounting for the formidable difference in proof times between MAX.COMMUTE and MAX.ASSOC.]

```
10_DD(NUMBER ABS(X)(IF (LESSP X 0) (MINUS X) X))
```

[The function ABS is now defined in the usual way.]

```
11_DF(FORALL.EXISTS (FORALL X (EXISTS Y (LESSEQP X Y))))
12_PR(FORALL.EXISTS)
Want instance for FORALL.EXISTS? Y
   Y/ (ABS X)
----------Proving----------
305 conses
.25 seconds
Proved
```

[The system asks for an instantiation of the existentially quantified variable Y. The user types in the instance term (ABS X). The instance is typechecked by the system and substituted for the variable Y in the Skolem form of the formula to be proved. The resulting ground formula is then proved by the underlying decision procedure.]

We now give as an example of the use of this system, the proof of the correspondance between the two most abstract levels in the design of the SIFT system [Sift:Agard]. This proof aims to demonstrate the validity of the design of SIFT by

- constructing a very abstract model of SIFT, simple enough to be evidently what is required by the users of the system. This description, in conventional mathematical notation, is simple enough to fit onto one page
- developing a hieirarchy of models of increasing complexity, culminating in the imperative Pascal program that implements the SIFT executive
- demonstrating that each of the axioms of each of these models can be proven as a theorem from the axioms of the model below it, though in many cases the axioms are identical and the 'proof' is trivial.

We include here the complete definitions of the most abstract model of SIFT, the IO Model, and of the next model of the SIFT hieirachy, the Replication Model. Also included are the set of lemmas, and the proofs of the lemmas, leading upto the proofs of the two most interesting axioms of the IO Model. Thses two axioms are the axioms stating that SIFT tasks get the correct results both when they are scheduled to execute and also when they are dormant. The proofs are, in effect, the proofs of the validity of majority voting to ensure correct operation of SIFT even in the presence of faults.

It is important to note that this example is a demonstration of the use of the system ON A REAL APPLICATION. Real applications turn out to be much bigger than the examples on which theorem provers are normally tested. Not only must the system accomodate models containing hundreds of axioms and lemmas but also the individual formulas can become very large. The more detailed levels of SIFT, where the theorem prover has also been successful, are yet more complex than the example we give here.

The IO Model

```
(
 (IEF INTEGER.STP)                          {These commands read into the
 (IEF SEQ.STP)                               system previously defined sets
 (IEF SETS.AXIOMS)                           of axioms}
 (IEF PAIROF.STP)

 (DTV TYPE1)                                {Type variable declatations}
 (DTV TYPE2)

 (DST REALTIME INTEGER)                     {Subtype declarations}
 (DST SUBFRAMETIME INTEGER)

 (DST INTERVAL (PAIR.OF SUBFRAMETIME SUBFRAMETIME))
 (DSV INTERVAL INTERVAL1)                   {A Varaible declaration}
 (DD SUBFRAMETIME BEGIN(INTERVAL1) (FIRST INTERVAL1))
 (DD SUBFRAMETIME END(INTERVAL1) (SECOND INTERVAL1))
                                            {Declarations of Definitions}
 (DD TYPE1 VALUE (PAIR1) (FIRST PAIR1))
 (DD TYPE2 SOURCE (PAIR1) (SECOND PAIR1))

 (DT FUNCTION.TYPE)                         {Declaration of an
                                             uninterpreted type}
 (DT SET.OF (TYPE1))

 (DST ITERATION INTEGER)
 (DSV ITERATION I)
 (DD ITERATION INCR(I) (IPLUS 1 I))

 (DT DATAVAL)
 (DST DATA (SEQ DATAVAL))
 (QUOTE "WAS (DT DATA)")
 (DT PROC)
 (DT TASK)

 (DSV TASK K)
 (DSV TASK L)
 (DS TASK GLOBAL.EXEC)                          {Declaration of a constant}
 (DS TASK CLOCK)
 (DS DATA BOTTOM1 (TASK))
 (DSV ITERATION J)
 (DSV SUBFRAMETIME T)
 (DSV SUBFRAMETIME TT)
 (DSV INTERVAL II)
 (DSV PROC P)
 (DSV PROC QQ)
 (DSV DATA V)
 (DSV (PAIR.OF DATA TASK)
      V.T)
 (DSV (SET.OF (PAIR.OF DATA TASK))
      V.INPUTS)
 (DSV (SET.OF (PAIR.OF DATA PROC))
      V.BAG)
```

```
(DS REALTIME EPSILON)
(DS REALTIME LAMBDA)
(DSV SUBFRAMETIME T1)
(DSV SUBFRAMETIME T2)
(DS INTERVAL OF (ITERATION TASK))                    {Declaration of Functions}
(DS INTERVAL DW.OF (ITERATION TASK))
(DS INTERVAL DW.FOR.TO.OF (TASK ITERATION TASK))
(DS ITERATION TO.OF (TASK ITERATION TASK))
(DS TASK ERROR.REPORTER (PROC))

(DSV SUBFRAMETIME T.SUB)
(DD SUBFRAMETIME SUB.INCR (T.SUB) (PLUS T.SUB 1))

(DD SUBFRAMETIME SUB.DECR (T.SUB) (DIFFERENCE T.SUB 1))

(DS TASK IC.ERROR.REPORTER (PROC))
(DS (SET.OF PROC)
    SAFE
    (SUBFRAMETIME))
(DS (SET.OF PROC)
    SAFE.FOR
    (INTERVAL))
(DS (SET.OF PROC)
    CONFIGURATION
    (DATA))
(DS BOOL TASK.SAFE (TASK ITERATION))
(DS (SET.OF PROC)
    POLL.FOR.OF
    (ITERATION TASK))
(DS (SET.OF DATA)
    ON
    (TASK ITERATION PROC))
(DS DATA ON.IN (TASK ITERATION PROC PROC))
(DS DATA IN (TASK ITERATION PROC))
(DS (SET.OF DATA)
    RESULT
    (TASK ITERATION))

(DS BOOL IC (TASK))
(DS BOOL ON.DURING (TASK ITERATION))
(DS BOOL SSF (TASK TASK))
(DS (SET.OF TASK)
    INPUTS
    (TASK))
(DS DATA APPLY (FUNCTION.TYPE (SET.OF (PAIR.OF DATA TASK))))
(DS FUNCTION.TYPE FUNCTION (TASK))
(DS REALTIME REAL.TIME (SUBFRAMETIME))
(DS BOOL REPORTS (PROC PROC ITERATION TASK))
(DS DATA REPORTVAL (PROC PROC ITERATION TASK))

(DS BOOL ON.DURING (TASK ITERATION))
(DS ITERATION TO.OF (TASK ITERATION TASK))
(DS BOOL TASK.SAFE (TASK ITERATION))
(DS (SET.OF DATA)
```

```
    RESULT
    (TASK ITERATION))
(DSV (PAIR.OF DATA PROC)
     V.P)

(DS BOOL IC.TASK.SAFE (TASK ITERATION))
(DS BOOL IC.TASK.SAFE (TASK ITERATION))
(DS TYPE1 SELECT ((SET.OF TYPE1)))



(DA IO.A1.1 (LESSP (SUB.INCR (BEGIN (OF I K)))
                   (END (OF I K))))

(DA IO.A1.2 (LESSEQP (END (OF I K))
                     (BEGIN (OF (INCR I)
                                K))))

(DA IO.A1.3 (IMPLIES (SSF L K)
                     (EQUAL (SUB.INCR (BEGIN (OF I K)))
                            (END (OF (TO.OF L I K)
                                     L)))))

(DA IO.A3 (IMPLIES (AND (IC K)
                        (IC.TASK.SAFE K I))
                   (EQUAL (CARD (RESULT K I))
                          1)))

(DA IO.A4 (IMPLIES (AND (IC K)
                        (MEMBER (SOURCE V.T)
                                (INPUTS K))
                        (SINGLETON V.INPUTS V.T))
                   (AND (EQUAL (CARD (INPUTS K))
                               1)
                        (IMPLIES (MEMBER L (INPUTS K))
                                 (EQUAL 1
                                        (CARD (POLL.FOR.OF (TO.OF L I K)
                                                           L))))
                        (EQUAL (VALUE V.T)
                               (APPLY (FUNCTION K)
                                      V.INPUTS)))))

(DA IO.A5 (IMPLIES (AND (MEMBER L (INPUTS K))
                        (ON.DURING K I)
                        (TASK.SAFE K I)
                        (NOT (ON.DURING L (TO.OF L I K))))
                   (AND (SINGLETON (RESULT L (TO.OF L I K))
                                   (BOTTOM1 L))
                        (TASK.SAFE L (TO.OF L I K)))))

(DA IO.A6 (IMPLIES (AND (LESSP T2 T1)
                        (FORALL I (IMPLIES (LESSEQP
                                              (END (OF I (CLOCK)))
                                              T1)
```

```
                                                    (TASK.SAFE (CLOCK)
                                                               I))))
                                (AND (LESSP (DIFFERENCE (TIMES (DIFFERENCE T1 T2)
                                                               (DIFFERENCE 1 (LAMBDA)))
                                                        (EPSILON))
                                            (DIFFERENCE (REAL.TIME T1)
                                                        (REAL.TIME T2)))
                                     (LESSP (DIFFERENCE (REAL.TIME T1)
                                                        (REAL.TIME T2))
                                            (PLUS (TIMES (DIFFERENCE T1 T2)
                                                         (PLUS 1 (LAMBDA)))
                                                  (EPSILON))))))

(DS (SET.OF (PAIR.OF DATA TASK)) V.INPUTS.A2 (ITERATION TASK))
(DA IO.A2A
    (IFF (AND (MEMBER (SOURCE V.T)
                      (INPUTS K))
              (MEMBER (VALUE V.T)
                      (RESULT (SOURCE V.T)
                              (TO.OF (SOURCE V.T)
                                     I K))))
         (MEMBER V.T (V.INPUTS.A2 I K))))


(DA IO.A2
    (IMPLIES (AND (ON.DURING K I)
                  (TASK.SAFE K I)
                  (FORALL L
                          (IMPLIES (MEMBER L (INPUTS K))
                                   (EQUAL (CARD (RESULT L
                                                        (TO.OF L I K)
                                                        ))
                                          1))))
             (SINGLETON (RESULT K I)
                        (APPLY (FUNCTION K) (V.INPUTS.A2 I K)))))

)
```

The Replication Model

```
(
 (IEF INTEGER.STP)
 (IEF SEQ.STP)
 (IEF SETS.AXIOMS)
 (IEF PAIROF.STP)

 (DTV TYPE2)
 (DTV TYPE1)
 (DST REALTIME INTEGER)

 (DST SUBFRAMETIME INTEGER)
 (DST INTERVAL (PAIR.OF SUBFRAMETIME SUBFRAMETIME))
 (DSV INTERVAL INTERVAL1)

 (DD SUBFRAMETIME BEGIN(INTERVAL1) (FIRST INTERVAL1))
 (DD SUBFRAMETIME END(INTERVAL1) (SECOND INTERVAL1))
 (QUOTE (DS SUBFRAMETIME BEGIN(INTERVAL)) )
 (QUOTE (DS SUBFRAMETIME END(INTERVAL)) )

 (DD TYPE1 VALUE (PAIR1) (FIRST PAIR1))
 (DD TYPE2 SOURCE (PAIR1) (SECOND PAIR1))

 (IEF MAJORITY.STP)

 (DT FUNCTION.TYPE)
 (DST ITERATION INTEGER)
 (DT DATAVAL)

 (DS DATAVAL BOTTOMD)
 (DSV DATAVAL D1)
 (DA BOTTOM.EQUALITY
     (EQUAL (BOTTOM D1) (BOTTOMD)))

 (DT TASK)
 (DSV TASK K)
 (DSV TASK L)
 (DS NAT RESULT.SIZE(TASK))


 (DST DATA (SEQ DATAVAL))
 (DSV DATA V)
 (DSV DATA V1)
 (DS DATA BOTTOM1 (TASK))
 (DA DATA.BOTTOM
     (IMPLIES
      (AND
       (LESSEQP 1 Y)
       (LESSEQP Y (RESULT.SIZE K)))
      (EQUAL (SEQ.ELEM (BOTTOM1 K) Y) (BOTTOMD))))
 (DA DATA.EQUALITY
     (IFF
      (EQUAL V V1)
```

```
      (FORALL Y
              (IMPLIES
               (AND
                (EQUAL (SEQ.LENGTH V) (SEQ.LENGTH V1))
                (LESSEQP 1 Y)
                (LESSEQP Y (SEQ.LENGTH V)))
               (EQUAL (SEQ.ELEM V Y) (SEQ.ELEM V1 Y))))))

(DT PROC)

(DS TASK GLOBAL.EXEC)
(DS TASK CLOCK)

(DSV ITERATION I)
(DSV ITERATION J)
(DSV ITERATION J1)
(DSV SUBFRAMETIME T)
(DSV SUBFRAMETIME TT)
(DSV INTERVAL II)
(DSV PROC P)
(DSV PROC QQ)
(DSV PROC R)
(DSV (PAIR.OF DATA TASK)
     V.T)
(DSV (SET.OF (PAIR.OF DATA TASK))
     V.INPUTS)
(DSV (SET.OF (PAIR.OF DATA PROC))
     V.BAG)
(DS REALTIME EPSILON)
(DS REALTIME LAMBDA)
(DSV SUBFRAMETIME T1)
(DSV SUBFRAMETIME T2)
(DS INTERVAL OF (ITERATION TASK))
(DS INTERVAL DW.OF (ITERATION TASK))
(DS INTERVAL DW.FOR.TO.OF (TASK ITERATION TASK))
(DS ITERATION TO.OF (TASK ITERATION TASK))
(DS TASK ERROR.REPORTER (PROC))

(DSV SUBFRAMETIME T.SUB)
(DD SUBFRAMETIME SUB.INCR (T.SUB) (PLUS T.SUB 1))

(DD SUBFRAMETIME SUB.DECR (T.SUB) (DIFFERENCE T.SUB 1))

(DS TASK IC.ERROR.REPORTER (PROC))
(DS (SET.OF PROC)  SAFE (SUBFRAMETIME))
(DS (SET.OF PROC)
    SAFE.FOR
    (INTERVAL))
(DS (SET.OF PROC)
    CONFIGURATION
    (DATA))
(DS BOOL TASK.SAFE (TASK ITERATION))
(DS (SET.OF PROC)
    POLL.FOR.OF
```

```
    (ITERATION TASK))
(DS (SET.OF DATA)
    ON
    (TASK ITERATION PROC))
(DS DATA ON.IN (TASK ITERATION PROC PROC))
(DS DATA IN (TASK ITERATION PROC))
(DS (SET.OF DATA)
    RESULT
    (TASK ITERATION))

(DS (SET.OF (PAIR.OF DATA TASK)) V.INPUTS.A2 (ITERATION TASK))
(DS DATA APPLY (FUNCTION.TYPE (SET.OF (PAIR.OF DATA TASK))))
(DS FUNCTION.TYPE FUNCTION (TASK))

(DA DATA.SIZE.IS.SEQ.LENGTH
    (AND
     (EQUAL (SEQ.LENGTH (IN K I QQ)) (RESULT.SIZE K))
     (EQUAL (SEQ.LENGTH (APPLY (FUNCTION K) (V.INPUTS.A2 I K)))
            (RESULT.SIZE K))
     (EQUAL (SEQ.LENGTH (BOTTOM1 K)) (RESULT.SIZE K))
     (EQUAL (SEQ.LENGTH (ON.IN K I P QQ)) (RESULT.SIZE K))))

(DA RESULT.SIZE.GREATER.THAN.1
    (GREATEREQP (RESULT.SIZE K) 1))

(DS BOOL IC (TASK))
(DS BOOL ON.DURING (TASK ITERATION))
(DS BOOL SSF (TASK TASK))
(DS (SET.OF TASK)
    INPUTS
    (TASK))

(DS REALTIME REAL.TIME (SUBFRAMETIME))
(DS BOOL REPORTS (PROC PROC ITERATION TASK))
(DS DATA REPORTVAL (PROC PROC ITERATION TASK))
(DS BOOL ON.DURING (TASK ITERATION))
(DS ITERATION TO.OF (TASK ITERATION TASK))
(DS BOOL TASK.SAFE (TASK ITERATION))
(DS (SET.OF DATA)
    RESULT
    (TASK ITERATION))

(DSV DATA V.CARD)
(DSV DATA V.CARD1)

(DSV (SET.OF TYPE1)
     S1)
(DSV (SET.OF TYPE1)
     S2)
(DSV DATA V2)
(DSV DATA V3)
(DSV DATA V4)
(DSV (PAIR.OF DATA PROC)
     V.P)
```

```
(DS (SET.OF (PAIR.OF DATAVAL PROC))  D.BAG.L10 (TASK ITERATION PROC NAT))

(DS BOOL IC.TASK.SAFE (TASK ITERATION))
(DS BOOL IC.TASK.SAFE (TASK ITERATION))
(DD ITERATION DECR(I)
    (DIFFERENCE I 1))
(DD ITERATION INCR (I)
    (IPLUS 1 I))



(DA RP.A1.1 (LESSP (SUB.INCR (BEGIN (OF I K)))
                   (END (OF I K))))
(DA RP.A1.2 (LESSEQP (END (OF I K))
                     (BEGIN (OF (INCR I)
                                K))))
(DA RP.A2 (IMPLIES (AND (MEMBER P (SAFE.FOR (OF I K)))
                        (MEMBER QQ (SAFE.FOR (OF I K))))
                   (SINGLETON (ON K I P)
                              (ON.IN K I P QQ))))
(DA RP.A7 (AND (EQUAL (CARD (INPUTS (ERROR.REPORTER P)))
                      0)
               (SINGLETON (INPUTS (IC.ERROR.REPORTER P))
                          (ERROR.REPORTER P))
               (IC (IC.ERROR.REPORTER P))
               (SINGLETON (POLL.FOR.OF I (ERROR.REPORTER P))
                          P)
               (EQUAL I (TO.OF (IC.ERROR.REPORTER P)
                               I
                               (ERROR.REPORTER P)))))

(DA RP.A8 (AND (MEMBER (IC.ERROR.REPORTER P)
                       (INPUTS (GLOBAL.EXEC)))
               (LESSP (BEGIN (OF I (GLOBAL.EXEC)))
                      (BEGIN (OF I (IC.ERROR.REPORTER QQ))))
               (LESSP (BEGIN (OF I (IC.ERROR.REPORTER QQ)))
                      (BEGIN (OF (INCR I)
                                 (GLOBAL.EXEC))))))

(DA RP.A9 (AND (SUBSET (CONFIGURATION (SELECT (RESULT (GLOBAL.EXEC)
                                                      I)))
                       (CONFIGURATION (SELECT (RESULT (GLOBAL.EXEC)
                                                      (DECR I)))))
               (IMPLIES (LESSP (END (OF I (GLOBAL.EXEC)))
                               (BEGIN (OF J K)))
                        (SUBSET (POLL.FOR.OF J K)
                                (CONFIGURATION (SELECT (RESULT (GLOBAL.EXEC)
                                                               I)))))))

(DA RP.D2.1 (EQUAL (BEGIN (DW.FOR.TO.OF L I K))
                   (IF (MEMBER L (INPUTS K))
                       (BEGIN (OF (TO.OF L I K)
                                  L))
```

```
                        (BEGIN (OF I K)))))

(DA RP.D2.2 (EQUAL (END (DW.FOR.TO.OF L I K))
                   (END (OF I K))))

(DA RP.D3.1 (NOT (LESSP (BEGIN (DW.FOR.TO.OF L I K))
                        (BEGIN (DW.OF I K)))))

(DA RP.D3.3 (EQUAL (END (DW.OF I K))
                   (END (OF I K))))

(DA RP.D7 (IFF (ON.DURING K I)
               (GREATERP (CARD (POLL.FOR.OF I K))
                         0)))


(DSV (PAIR.OF DATAVAL PROC) D.P)
(DSV (SET.OF (PAIR.OF DATAVAL PROC)) D.BAG)
(DS (SET.OF (PAIR.OF DATAVAL PROC)) D.BAG.D4 (TASK ITERATION PROC NAT))

(DA RP.D4A
    (IFF (MEMBER D.P (D.BAG.D4 K I QQ Y))
         (EXISTS P (AND (EQUAL (SEQ.ELEM (ON.IN K I P QQ) Y)
                               (VALUE D.P))
                        (EQUAL P (SOURCE D.P))
                        (MEMBER P (POLL.FOR.OF I K))))))

(DA RP.D4 (IMPLIES
           (AND
            (MEMBER QQ (SAFE.FOR (OF I K)))
            (LESSEQP 1 Y)
            (LESSEQP Y (RESULT.SIZE K)) )
           (EQUAL (SEQ.ELEM (IN K I QQ) Y)
                  (MAJORITY (D.BAG.D4 K I QQ Y)))))

(DS (SET.OF (PAIR.OF DATA TASK)) V.INPUTS.A3 (TASK ITERATION PROC))
(DA RP.A3A
    (IFF (MEMBER V.T (V.INPUTS.A3 K I P))
         (AND (MEMBER (SOURCE V.T)
                      (INPUTS K))
              (EQUAL (VALUE V.T)
                     (IN (SOURCE V.T)
                         (TO.OF (SOURCE V.T)
                                I K)
                         P)))))

(DA RP.A3 (IMPLIES
           (MEMBER P (INTERSECTION (POLL.FOR.OF I K)
                                   (SAFE.FOR (DW.OF I K))))
           (SINGLETON (ON K I P)
                      (APPLY (FUNCTION K) (V.INPUTS.A3 K I P)))))

(DA RP.D1
    (AND (IMPLIES (AND (MEMBER L (INPUTS K))
```

```
                    (NOT (SSF L K)))
                (AND (NOT (LESSP (BEGIN (OF I K))
                                 (END (OF (TO.OF L I K)
                                          L))))
                     (LESSP (BEGIN (OF I K))
                            (END (OF (INCR (TO.OF L I K))
                                     L)))))
        (IMPLIES (AND (MEMBER L (INPUTS K))
                      (SSF L K))
                 (EQUAL (END (OF (TO.OF L I K)
                                 L))
                        (SUB.INCR (BEGIN (OF I K)))))))

(DA IO.D1
    (AND (IMPLIES (AND (MEMBER L (INPUTS K))
                       (NOT (SSF L K)))
                  (AND (NOT (LESSP (BEGIN (OF I K))
                                   (END (OF (TO.OF L I K)
                                            L))))
                       (LESSP (BEGIN (OF I K))
                              (END (OF (INCR (TO.OF L I K))
                                       L)))))
         (IMPLIES (AND (MEMBER L (INPUTS K))
                       (SSF L K))
                  (EQUAL (END (OF (TO.OF L I K)
                                  L))
                         (SUB.INCR (BEGIN (OF I K)))))))


(DA RP.D11
    (IFF (MEMBER D.P (D.BAG.L10 K I QQ Y))
         (EXISTS P
                 (AND
                  (MEMBER P (INTERSECTION (POLL.FOR.OF I K)
                                          (SAFE.FOR (DW.OF I K))))
                  (EQUAL (SEQ.ELEM (ON.IN K I P QQ) Y)
                         (VALUE D.P))
                  (EQUAL P (SOURCE D.P))))))


(DA RP.D9A (IFF (TASK.SAFE K I)
                (OR (NOT (ON.DURING K I))
                    (LESSP (CARD (POLL.FOR.OF I K))
                           (TIMES 2
                                  (CARD (INTERSECTION (POLL.FOR.OF I K)
                                                      (SAFE.FOR
                                                       (DW.OF I K)))))))))
(DA RP.A4 (IMPLIES (AND (IC K)
                        (IC.TASK.SAFE K I))
                   (EQUAL (CARD (RESULT K I))
                          1)))
(DA RP.D9B
  (IFF
    (IC.TASK.SAFE K I)
```

```
    (OR (NOT (ON.DURING K I))
        (AND (IC K)
             (IMPLIES (MEMBER L (INPUTS K))
                      (LESSP (TIMES 2
                                    (CARD (UNION (POLL.FOR.OF (TO.OF L I K)
                                                              L)
                                                 (POLL.FOR.OF I K))))
                             (TIMES 3
                                    (CARD (INTERSECTION
                                            (SAFE.FOR (DW.OF I K))
                                            (UNION (POLL.FOR.OF (TO.OF L I K)
                                                                L)
                                                   (POLL.FOR.OF I K)))))))))))


(DA RP.A5 (IMPLIES (AND (IC K)
                        (MEMBER (SOURCE V.T)
                                (INPUTS K))
                        (SINGLETON V.INPUTS V.T))
                   (AND (EQUAL (CARD (INPUTS K))
                               1)
                        (IMPLIES (MEMBER L (INPUTS K))
                                 (EQUAL 1
                                        (CARD (POLL.FOR.OF (TO.OF L I K)
                                                           L))))
                        (EQUAL (VALUE V.T)
                               (APPLY (FUNCTION K)
                                      V.INPUTS)))))

(DA RP.A10 (IMPLIES (MEMBER P (SAFE.FOR (DW.OF J K)))
                    (IFF (AND (NOT (IC L))
                              (MEMBER L (INPUTS K))
                              (MEMBER QQ (POLL.FOR.OF (TO.OF L J K)
                                                      L))
                              (NOT (EQUAL (ON.IN L (TO.OF L J K)
                                                 QQ P)
                                          (IN L (TO.OF L J K)
                                              P))))
                         (REPORTS P QQ (TO.OF L J K)
                                  L))))

(DF RP.A11 (IMPLIES (GREATERP T1 T2)
                    (SUBSET (SAFE T1)
                            (SAFE T2))))


(DA RP.A6 (IMPLIES (AND (LESSP T2 T1)
                        (FORALL I (IMPLIES (LESSEQP
                                             (END (OF I (CLOCK)))
                                             T1)
                                           (TASK.SAFE (CLOCK)
                                                      I))))
                   (AND (LESSP (DIFFERENCE (TIMES (DIFFERENCE T1 T2)
```

```
                                                    (DIFFERENCE 1 (LAMBDA)))
                                                (EPSILON))
                                        (DIFFERENCE (REAL.TIME T1)
                                                    (REAL.TIME T2)))
                                (LESSP (DIFFERENCE (REAL.TIME T1)
                                                   (REAL.TIME T2))
                                       (PLUS (TIMES (DIFFERENCE T1 T2)
                                                    (PLUS 1 (LAMBDA)))
                                             (EPSILON))))))

(DA RP.D3.2 (EXISTS L (EQUAL (BEGIN (DW.FOR.TO.OF L I K))
                             (BEGIN (DW.OF I K)))))

(DA RP.D8 (IFF (REPORTS P QQ I K)
               (EXISTS J
                       (AND (LESSEQP (BEGIN (OF I K))
                                     (BEGIN (OF J (ERROR.REPORTER P))))
                            (LESSP (BEGIN (OF (DECR J)
                                              (ERROR.REPORTER P)))
                                   (END (OF (TO.OF L I K)
                                            L)))
                            (MEMBER (REPORTVAL P QQ I K)
                                    (RESULT (ERROR.REPORTER P)
                                            J))))))

(DA RP.D10 (IFF (FORALL T (IMPLIES (AND (LESSEQP (BEGIN II)
                                                 T)
                                        (LESSP T (END II)))
                                   (MEMBER P (SAFE T))))
                (MEMBER P (SAFE.FOR II))))


(DA RP.D6
    (IFF
     (MEMBER V (RESULT K I))
     (EXISTS P (AND
                (MEMBER P (SAFE.FOR (OF I K)))
                (EQUAL V (IN K I P))))))

)
```

The Lemmas

```
(
(QUOTE "The begining of a Data Window is earlier or at least equal to ne
       begining of the Execution Window" )
(DF RP.L1 (GREATEREQP (BEGIN (OF I K))
                      (BEGIN (DW.FOR.TO.OF L I K))))

(QUOTE "If a time is within the Execution Window, then it must be within
       the Data Window")
(DF RP.L2 (IMPLIES (AND (LESSEQP (BEGIN (OF I K))
                                 T)
                        (LESSP T (END (OF I K))))
                   (AND (LESSEQP (BEGIN (DW.OF I K))
                                 T)
                        (LESSP T (END (DW.OF I K))))))

(QUOTE "If a processor is Safe for the Data Window, it is Safe for the
       Execution Window")
(DF RP.L3 (IMPLIES (MEMBER P (SAFE.FOR (DW.OF I K)))
                   (MEMBER P (SAFE.FOR (OF I K)))))

(QUOTE "If a task generates a singleton result value, then safe processors
       will have that value in their In buffer")
(DF RP.L4 (IMPLIES (AND (MEMBER L (INPUTS K))
                        (EQUAL 1 (CARD (RESULT L (TO.OF L I K))))
                        (MEMBER P (SAFE.FOR (DW.OF I K))))
                   (IFF (MEMBER V (RESULT L (TO.OF L I K)))
                        (EQUAL V (IN L (TO.OF L I K)
                                       P)))))

(QUOTE "If a task is on a processor that is Safe for its data window,
       and if all its input tasks are well behaved, the inputs to the
       task will be same as in the IO Model")
(DF RP.L5 (IMPLIES (AND
                    (FORALL L
                         (IMPLIES
                          (MEMBER L (INPUTS K))
                          (EQUAL 1 (CARD (RESULT L (TO.OF L I K))))))
                    (MEMBER P (SAFE.FOR (DW.OF I K))))
                   (IFF (MEMBER V.T (V.INPUTS.A3 K I P))
                        (MEMBER V.T (V.INPUTS.A2 I K)))))

(QUOTE "As RP.L5")
(DF RP.L6 (IMPLIES
           (AND
            (FORALL L
                     (IMPLIES
                      (MEMBER L (INPUTS K))
                      (EQUAL 1 (CARD (RESULT L (TO.OF L I K))))))
            (MEMBER P (SAFE.FOR (DW.OF I K))))
           (EQUAL (V.INPUTS.A2 I K) (V.INPUTS.A3 K I P))))

(QUOTE "If a processor is Safe for the Data Window of a task, it is Safe for
```

```
        the Execution Windows of each of that task's input tasks.  Needed to
        prove RP.L4")
(DF RP.L7 (IMPLIES (AND (MEMBER P (SAFE.FOR (DW.OF I K)))
                        (MEMBER L (INPUTS K)))
                   (MEMBER P (SAFE.FOR (OF (TO.OF L I K)
                                           L)))))

(QUOTE "If a processor executes a task, and is Safe for the data wwindow of
        that task, and if all the inputs to the task are well behaved, then
        the the task output computed by that processor will be the result of
        applying the task function to the correct task inputs")
(DF RP.L8 (IMPLIES
           (AND
            (MEMBER P (INTERSECTION (POLL.FOR.OF I K)
                                    (SAFE.FOR (DW.OF I K))))
            (FORALL L
                    (IMPLIES
                     (MEMBER L (INPUTS K))
                     (EQUAL 1 (CARD (RESULT L (TO.OF L I K)))))))
                   (SINGLETON (ON K I P)
                              (APPLY (FUNCTION K)
                                     (V.INPUTS.A2 I K)))))

(QUOTE "... and that output value will be the broadcast value received by
        all processors that are Safe for the execution window of the task")
(DF RP.L9 (IMPLIES (AND (MEMBER P (INTERSECTION (POLL.FOR.OF I K)
                                                (SAFE.FOR (DW.OF I K))))
                        (FORALL L
                                (IMPLIES
                                 (MEMBER L (INPUTS K))
                                 (EQUAL 1 (CARD (RESULT L (TO.OF L I K))))))
                        (MEMBER QQ (SAFE.FOR (OF I K))))
                   (EQUAL (ON.IN K I P QQ)
                          (APPLY (FUNCTION K)
                                 (V.INPUTS.A2 I K)))))

(QUOTE "A result value received from a Safe processor is a member of the set
        of all computer result values")
(DF RP.L11 (IMPLIES (AND (MEMBER QQ (SAFE.FOR (OF I K)))
                         (MEMBER D.P (D.BAG.L10 K I QQ Y)))
                    (MEMBER D.P (D.BAG.D4 K I QQ Y))))

(DSV (PAIR.OF DATAVAL PROC) D.P.1)
(DS (SET.OF (PAIR.OF DATAVAL PROC)) D.BAG.L12
    (TASK ITERATION PROC NAT))


(QUOTE "Definition of D.BAG.L12 to be the set of correct values in the set of
        result values to be voted on")
(DA RP.L12A
   (IFF
    (MEMBER D.P.1 (D.BAG.L12 K I QQ Y))
    (AND
     (EQUAL
```

```
      (SEQ.ELEM (APPLY (FUNCTION K) (V.INPUTS.A2 I K))
                Y)
      (VALUE D.P.1))
     (MEMBER D.P.1 (D.BAG.D4 K I QQ Y)))))

(QUOTE "If a processor is Safe for the execution window of a task, and that
        generates a singleton result value, then the result values received
        from Safe processors by that processor will be correct values")
(DF RP.L12R (IMPLIES (AND (MEMBER QQ (SAFE.FOR (OF I K)))
                          (MEMBER D.P (D.BAG.L10 K I QQ Y))
                          (FORALL L
                                  (IMPLIES
                                   (MEMBER L (INPUTS K))
                                   (EQUAL 1 (CARD (RESULT L (TO.OF L I K)))))
                    (MEMBER D.P (D.BAG.L12 K I QQ Y))))

(QUOTE "as RP.L12R but as subset")
(DF RP.L13 (IMPLIES
            (AND
             (MEMBER QQ (SAFE.FOR (OF I K)))
             (FORALL L
                     (IMPLIES
                      (MEMBER L (INPUTS K))
                      (EQUAL 1 (CARD (RESULT L (TO.OF L I K)))))))
            (SUBSET (D.BAG.L10 K I QQ Y) (D.BAG.L12 K I QQ Y))))

(QUOTE "A time within the execution window of an input task to a task K
        lies within the data window of task K. Used to prove RP.L7")
(DF RP.L2A (IMPLIES (AND (LESSEQP (BEGIN (OF (TO.OF L I K)
                                             L))
                                  T)
                         (LESSP T (END (OF (TO.OF L I K)
                                           L)))
                         (MEMBER L (INPUTS K)))
                    (AND (LESSEQP (BEGIN (DW.OF I K))
                                  T)
                         (LESSP T (END (DW.OF I K))))))

(QUOTE "If a task executes and is Safe, at least one processor must have been
        Safe for its execution window")
(DF RP.L16 (IMPLIES (AND (ON.DURING K I)
                         (TASK.SAFE K I))
                    (GREATERP (CARD (SAFE.FOR (OF I K)))
                              0)))

(QUOTE "A Primary Lemma.  If a task executes and is Safe, and if all its
        inputs are well behaved, a Safe processor voting on the broadcast
        results will obtain the correct result value for that task")
(DF RP.L14 (IMPLIES (AND (TASK.SAFE K I)
                         (ON.DURING K I)
                         (MEMBER QQ (SAFE.FOR (OF I K)))
                         (FORALL L
                            (IMPLIES
                             (MEMBER L (INPUTS K))
```

```
                                (EQUAL 1 (CARD (RESULT L (TO.OF L I K))))))
                            (LESSEQP 1 Y)
                            (LESSEQP Y (RESULT.SIZE K)))
                       (EQUAL (SEQ.ELEM (APPLY (FUNCTION K)
                                               (V.INPUTS.A2 I K)) Y)
                              (MAJORITY (D.BAG.D4 K I QQ Y)))))

(QUOTE "... and will place that result value in its IN buffer")
(DF RP.L15 (IMPLIES (AND (TASK.SAFE K I)
                         (ON.DURING K I)
                         (MEMBER QQ (SAFE.FOR (OF I K)))
                         (FORALL L
                            (IMPLIES
                             (MEMBER L (INPUTS K))
                             (EQUAL 1 (CARD (RESULT L (TO.OF L I K)))))))
                    (EQUAL (APPLY (FUNCTION K)
                                  (V.INPUTS.A2 I K))
                           (IN K I QQ))))

(QUOTE "Almost there:  If a task executes and is Safe, and all its inputs
        are well behaved, its result will be the result of applying its
        function to the correct inputs")
(DSV TASK L1)
(DF RP.L17
    (IMPLIES (AND (TASK.SAFE K I)
                  (ON.DURING K I)
                  (FORALL L
                          (IMPLIES
                           (MEMBER L (INPUTS K))
                           (EQUAL 1 (CARD (RESULT L (TO.OF L I K))))))
                  (FORALL L1
                          (IMPLIES
                           (MEMBER L1 (INPUTS K))
                           (EQUAL 1 (CARD (RESULT L1 (TO.OF L1 I K)))))))
             (SINGLETON (RESULT K I)
                        (APPLY (FUNCTION K)
                               (V.INPUTS.A2 I K)))))

(QUOTE "The number of versions of a task's result available for voting on is
        the number of processors executing that task")
(DF CARD.D.BAG.D4
     (EQUAL (CARD (D.BAG.D4 K I QQ Y))
            (CARD (POLL.FOR.OF I K))))

(QUOTE "The number of correct versions of a task's result is the number of
        Safe processors executing that task")
(DF CARD.D.BAG.L10 (EQUAL (CARD (D.BAG.L10 K I QQ Y))
                          (CARD (INTERSECTION (POLL.FOR.OF I K)
                                              (SAFE.FOR (DW.OF I K))))))

(DSV TASK L2)
(DF NECESSARY.EVIL
   (IMPLIES
     (FORALL L
```

```
            (IMPLIES
             (MEMBER L (INPUTS K))
             (EQUAL 1 (CARD (RESULT L (TO.OF L I K))))))
      (AND
       (FORALL L1
            (IMPLIES
             (MEMBER L1 (INPUTS K))
             (EQUAL 1 (CARD (RESULT L1 (TO.OF L1 I K))))))
       (FORALL L2
            (IMPLIES
             (MEMBER L2 (INPUTS K))
             (EQUAL 1 (CARD (RESULT L2 (TO.OF L2 I K)))))))))

(QUOTE "We now consider tasks that are not currently being executed")

(QUOTE "If a task is executed and safe, and has an input task that is not
        being executed, a majority of the result values for that not.on task
        will be nulls")
(DF RP.L19 (IMPLIES (AND (MEMBER L (INPUTS K))
                         (ON.DURING K I)
                         (TASK.SAFE K I)
                         (NOT (ON.DURING L (TO.OF L I K)))
                         (LESSEQP 1 Y)
                         (LESSEQP Y (RESULT.SIZE L)))
                    (EQUAL (BOTTOMD)
                           (MAJORITY (D.BAG.D4 L (TO.OF L I K)
                                               QQ Y)))))

(QUOTE "... and on a safe processor that null value will be placed in the
        IN buffer")
(DF RP.L20 (IMPLIES (AND (MEMBER L (INPUTS K))
                         (ON.DURING K I)
                         (TASK.SAFE K I)
                         (NOT (ON.DURING L (TO.OF L I K)))
                         (MEMBER QQ (SAFE.FOR (OF (TO.OF L I K) L)))
                         (LESSEQP 1 Y)
                         (LESSEQP Y (RESULT.SIZE L)))
                    (EQUAL (SEQ.ELEM (BOTTOM1 L) Y)
                           (SEQ.ELEM (IN L (TO.OF L I K) QQ) Y))))

(QUOTE "as RP.L20")
(DF RP.L21 (IMPLIES (AND (MEMBER L (INPUTS K))
                         (ON.DURING K I)
                         (TASK.SAFE K I)
                         (NOT (ON.DURING L (TO.OF L I K)))
                         (MEMBER QQ (SAFE.FOR (OF (TO.OF L I K) L))))
                    (EQUAL (BOTTOM1 L)
                           (IN L (TO.OF L I K) QQ))))


)
```

The Proof Commands with the required Instantiations

```
(
(PR (RP.L1)
    (RP.A1.1
     ((K L)
      (I (TO.OF L I K))))
    (RP.D1)
    (RP.D2.1))

(PR (RP.L2)
    (RP.A1.1)
    (RP.D3.3)
    (RP.D3.1)
    (RP.L1))

(PR (RP.L2A)
    (RP.A1.1)
    (RP.D3.3)
    (RP.D3.1)
    (RP.D2.1)
    (RP.D1))

(PR (RP.L3)
    (RP.L2
     ((T *T:3)))
    (RP.D10
     ((T *T:3)
      (II (DW.OF I K))))
    (RP.D10
     ((II (OF I K))
      (T D))))

(PR (RP.L7)
    (RP.D10
     ((T *T:1)
      (II (OF (TO.OF L I K) L))))
    (RP.D10
     ((T *T:1)
      (II (DW.OF I K))))
    (RP.L2A
     ((T *T:1))))

(PR (RP.L16)
    (CARD.INTERSECTION
     ((S1 (SAFE.FOR (DW.OF I K)))
      (S (POLL.FOR.OF I K))))
    (CARD.SUBSET
     ((S2 (SAFE.FOR (OF I K)))
      (S1 (SAFE.FOR (DW.OF I K)))))
    (SUBSET
     ((S2 (SAFE.FOR (OF I K)))
      (X *X:3)
      (S1 (SAFE.FOR (DW.OF I K))))))
```

```
      (RP.L3
       ((P *X:3)))
      (RP.D9A)
      (RP.D7))

 (PR (RP.L4)
     (RP.L7)
     (CARD.2
      ((X (IN L (TO.OF L I K) P))
       (X1 V)
       (S (RESULT L (TO.OF L I K)))))
     (RP.D6
      ((I (TO.OF L I K))
       (K L)
       (V (IN L (TO.OF L I K) P))))
     (RP.D6
      ((I (TO.OF L I K))
       (K L))))

 (PR (RP.L5
       ((L (SOURCE V.T))))
     (RP.A3A)
     (IO.A2A)
     (RP.L4
      ((V (VALUE V.T))
       (L (SOURCE V.T)))))

 (PR (RP.L6
       ((L *L:2)))
     (SETEQUALITY
      ((S2 (V.INPUTS.A3 K I P))
       (S1 (V.INPUTS.A2 I K))
       (X *X:1)))
     (RP.L5
      ((V.T *X:1))))

 (PR (RP.L8
      ((L *L:2)))
     (INTERSECT
      ((S1 (SAFE.FOR (DW.OF I K)))
       (S (POLL.FOR.OF I K))
       (X P)))
     (RP.L6)
     (RP.A3))

 (PR (RP.L9
      ((L *L:5)))
     (INTERSECT
      ((S1 (SAFE.FOR (DW.OF I K)))
       (S (POLL.FOR.OF I K))
       (X P)))
     (CARD.2
      ((X1 (ON.IN K I P QQ))
       (X (APPLY (FUNCTION K) (V.INPUTS.A2 I K)))
```

```
      (S (ON K I P))))
    (RP.L3)
    (RP.A2)
    (RP.L8))

(PR (RP.L11)
    (INTERSECT
     ((S1 (SAFE.FOR (DW.OF I K)))
      (S (POLL.FOR.OF I K))
      (X *P:2)))
     (RP.D11
      ((P D)))
     (RP.D4A
      ((P *P:2))))

(PR (RP.L12R
     ((L *L:3)))
    (RP.L12A
     ((D.P.1 D.P)))
    (RP.L11)
    (RP.L9
     ((P *P:4)))
    (RP.D11
     ((P D))))

(PR (RP.L13
     ((L *L:2)))
    (SUBSET
     ((S2 (D.BAG.L12 K I QQ Y))
      (X *X:1)
      (S1 (D.BAG.L10 K I QQ Y))))
    (RP.L12R
     ((D.P *X:1))))

(PR (RP.L14
     ((L *L:1)))
    (RP.L13)
    (RP.D9A)
    (CARD.D.BAG.D4)
    (CARD.D.BAG.L10)
    (CARD.SUBSET
     ((S2 (D.BAG.L12 K I QQ Y))
      (S1 (D.BAG.L10 K I QQ Y))))
    (MAJ.1
     ((T1.V (SEQ.ELEM (APPLY (FUNCTION K) (V.INPUTS.A2 I K)) Y))
      (M.BAG.1 (D.BAG.L12 K I QQ Y))
      (M.BAG (D.BAG.D4 K I QQ Y))))
    (RP.L12A
     ((D.P.1 *V1.V2:6))))

(PR (RP.L15
     ((L *L:1)))
    (RP.L14
     ((Y *Y:3)))
```

```
    (RP.D4
     ((Y *Y:3)))
    (DATA.EQUALITY
     ((V (IN K I QQ))
      (Y D)
      (V1 (APPLY (FUNCTION K) (V.INPUTS.A2 I K))) ))
    (DATA.SIZE.IS.SEQ.LENGTH)
    (RESULT.SIZE.GREATER.THAN.1)) )

(PR (RP.L17
     ((L *L:6)
      (I 1 *L:7)))
    (CARD.3
     ((V.CARD.3 (APPLY (FUNCTION K) (V.INPUTS.A2 I K)))
      (S (RESULT K I))))
    (RP.L16)
    (CARD.4
     ((S (SAFE.FOR (OF I K)))))
    (RP.D6
     ((P *X:3)
      (V (APPLY (FUNCTION K) (V.INPUTS.A2 I K)))))
    (RP.D6
     ((V *X:1)
      (P D)))
    (RP.L15
     ((QQ *X:3)))
    (RP.L15
     ((QQ *P:5))))

(PR (IO.A2
     ((L *L:2)))
    (RP.L17)
    (NECESSARY.EVIL
     ((L1 *L:1)
      (L2 *L1:1))))


(PR (RP.L18)
    (CARD.4
     ((S (SAFE.FOR (DW.OF I K)))))
    (RP.L7
     ((P (*X.CARD.4 (SAFE.FOR (DW.OF I K))))))
    (CARD.INTERSECTION
     ((S (POLL.FOR.OF I K))
      (S1 (SAFE.FOR (DW.OF I K)))))
   (RP.D9A)
    (RP.D7))

(PR (RP.L19)
    (MAJ.2
     ((M.BAG (D.BAG.D4 L (TO.OF L I K) QQ Y))
      (T2.V D1:2)))
    (BOTTOM.EQUALITY)
     (CARD.D.BAG.D4
```

```
      ((K L)
       (I (TO.OF L I K))))
     (CARD.6
      ((S (POLL.FOR.OF (TO.OF L I K) L))))
     (RP.D7
      ((K L)
       (I (TO.OF L I K)))))

(PR (RP.L20)
    (RP.L19)
    (DATA.BOTTOM
     ((K L)))
    (DATA.EQUALITY
     ((V (IN L (TO.OF L I K) QQ))
      (V1 (BOTTOM1 L))))
   (RP.D4
    ((K L)
      (I (TO.OF L I K)))))

(PR (RP.L21)
    (RP.L20
     ((Y *Y:2)))
    (DATA.EQUALITY
     ((V (IN L (TO.OF L I K) QQ))
      (Y D)
      (V1 (BOTTOM1 L))))
    (DATA.SIZE.IS.SEQ.LENGTH
     ((K L)
      (I (TO.OF L I K)))))


(PR (IO.A5)
    (RP.D9A
     ((K L)
      (I (TO.OF L I K))))
    (RP.L21
     ((QQ *X:9)))
    (RP.L18)
    (RP.L21
     ((QQ *P:6)))
    (RP.D6
     ((K L)
      (I (TO.OF L I K))
      (V (BOTTOM1 L))
      (P *X:9 )))
    (RP.D6
     ((K L)
      (I (TO.OF L I K))
      (P D)
      (V *X:7)))
    (CARD.3
     ((V.CARD.3 (BOTTOM1 L))
      (S (RESULT L (TO.OF L I K)))))
    (RP.L21
```

```
     ((QQ *X:9)))

    (CARD.4
     ((S (SAFE.FOR (DW.OF I K)))))
    (RP.L7
     ((P *X:9)))
    (CARD.INTERSECTION
     ((S (POLL.FOR.OF I K))
      (S1 (SAFE.FOR (DW.OF I K)))))
   (RP.D9A)
    (RP.D7))

)
```

IV

Some Completeness Results for a Class of Inequality Provers

by

W. W. Bledsoe, Robert Neveln and Robert Shostak

Abstract. A modified resolution procedure, RCF, which uses a restricted form of inequality chaining and variable elimination is proved to be complete, for first order logic. RCF allows chaining only on terms of the form $f(t_1, \ldots, t_n)$ where f is an uninstantiated function symbol and $n \geq 1$. (E.g., we never chain on variables.) Other results are given. A prover using RCS+, an extension of RCF, has been implemented and used to prove several moderately difficult inequality theorems, not proved earlier by general purpose automatic provers.

1. Introduction

One of the most effective procedures used in our inequality prover [1] is that of variable elimination, whereby a variable which is "eligible" (see below) in a clause, can be eliminated from that clause. For example, the clause

(1) $\quad a \not\leq x \vee x \not\leq b \vee c \leq d$

can be replaced by the clause

(1') $\quad a \not\leq b \vee c \leq d$

by elimination of the variable x (assuming that x does not occur in a,b,c, or d). Also, the variable x (which does not occur in a,b, or c) can be eliminated from the clause

(2) $\quad a \not\leq x \vee b \leq c$

to produce the clause

(2') $\quad b \leq c$.

In general, the variable x (which does not occur in a_i, b_j, or E) can be eliminated from the clause

$$\left(\bigvee_{i=1}^{n} a_i \not\leq x \vee \bigvee_{j=1}^{m} x \not\leq b_j \vee E\right)$$

to produce

$$\left(\bigvee_{i=1}^{n}\bigvee_{j=1}^{m} a_i \not\leq b_j \vee E\right) .$$

A variable is <u>eligible</u> in a clause if it does not occur within the arguments of an uninstantiated function symbol. Thus x is eligible in (1) but not in (3).

(3) $\quad a \not\leq x \vee x \not\leq b \vee f(x) \leq c ,$

because it occurs as an argument of the uninstantiated function symbol f. The term $f(x)$ is called a <u>shielding term</u> because it "shields" the variable x, thereby preventing it from being eligible in (3).

The principal objective of the inequality prover [1] is to remove such shielding terms, by inequality "chaining" and other procedures (see below), so that variables <u>can</u> be eliminated.

The clause

$$R = (a \leq c \vee E_1 \vee E_2)\sigma$$

is said to be a <u>chain-resolvent</u> of clauses

$$C_1 = (a \leq b \vee E_1) ,$$

and

$$C_2 = (b' \leq c \vee E_2) ,$$

if σ is the Mgu of $\{b,b'\}$. We also allow "self-chaining" whereby $E\sigma$ is inferred from $(b < b' \vee E)$.

We will designate by RC ("resolution chaining") a procedure which only uses chaining (as described above) and factoring. RC was shown to be complete by Slagle [2,3] (See also Lemma 4, Section 3.) Unfortunately RC alone is not very powerful as a prover. In order to strengthen RC, we have added VE (variable elimination, as described above), and have imposed restrictions on the chaining process, which help control proof search tree.

Two such procedures are RCF and RCS, which are described as follows. Both RCF and RCS use VE, and both restrict chaining as follows: Let

$$R = (a \leq c \vee E_1 \vee E_2)\sigma$$

be the chain resolvent of

$$C_1 = (a \leq b \vee E_1) \quad \text{and} \quad C_2 = (b' \leq c \vee E_2) ,$$

where $\sigma = \text{Mgu}(b,b')$. We accept R as an RCF chain resolvent if

(1) all of a,b,b',c are ground terms (and hence $b = b'$), or

(2) b and b' are both of the form $f(t_1, \ldots, t_n)$ where f is an uninstantiated function symbol, and $n \geq 1$.

And we accept R as an RCS chain resolvent, if additionally, in case (2), either b or is non-ground, i.e., either b or b' is a shielding term.

Other restrictions on RC include RCM and RC+. RCM uses "multiple cuts", where, for example, two clauses

$$C_1 = (a \leq c \vee b \leq c \vee E_1)$$

and

$$C_2 = (c \leq d \vee c \leq e \vee E_2)$$

are chained, in one step, on both c's in C_1 and both c's in C_2 to obtain

$$(a \leq d \vee a \leq e \vee b \leq d \vee b \leq e \vee E_1 \vee E_2) .$$

RC+ permits literals of the form

$$a_1 + \ldots + a_n \leq b_1 + \ldots + b_m ,$$

where the a_i and b_j are traditional terms (with no occurrence of +). Two such literals are chained by cancelling like terms (after unification). For example,

$$f(x) + a \leq h(x)$$

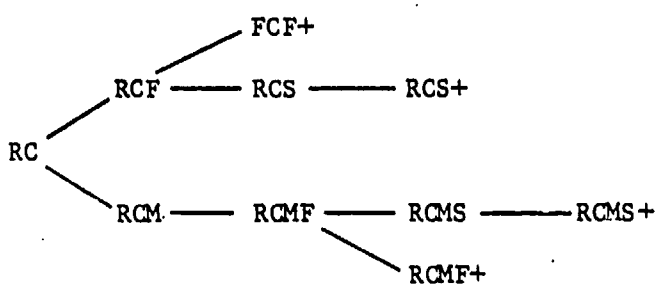
and

$$b \leq f(c)$$

are RC+ chained to obtain

$$b + a \leq h(c) .$$

By combining these restrictions we obtain the following diagram



where more restrictive (stronger) procedures are shown to the right.

It is the purpose of this paper to prove that RCF, RCF+, RCM, RCMF, and RCMF+ are complete.

It is conjectured that RCS is also complete, as well as RCS+, RCMS, and RCMS+.

RCS+ is the procedure described in [1]. But RCF+, which is proved complete here, is equally as strong as RCS on the examples given in [1]. Since we allow quantification and uninterpreted function symbols, we can encode all of first order logic. For example, the atom $P(x,y)$ can be written as

$$f(x,y) \leq 0$$

where f is a new uninterpreted function symbol associated with P. Hence our procedures RC, RCF, etc. are complete for all of first order logic.

In each of RCF, RCS, RCM, etc., it is required that variable elimination (VE) be applied immediately when a variable becomes eligible in a clause C, and that C be discarded and replaced by its VE-resolvent.

The reader might prefer to skip to Section 3, page 19, and refer back to Section 2 as needed.

2. Definitions and Logical Basis

2.1. Axioms for total (linear) order: T

1. $x \not< x$ — Anti-reflexive
2. $x < y \to y \not< x$ — Anti-symmetry
3. $x < y \wedge y < z \to x < z$ — Transitivity
4. $y \not< x \wedge z \not< y \to z \not< x$ — "

It is convenient (but not necessary) to also use the symbol "$\leq$", where $a \leq b$ is equivalent to $b \not< a$. Then axioms 1-4 can be written

1. $x \leq x$
2. $x < y \to x \leq y$
3. $x < y \wedge y < z \to x < z$
4. $x \leq y \wedge y \leq z \to x \leq z$

The axioms of 1-4 are also called the inequality axioms.

Definition. Let $S_{\leq}$ be the set of clauses corresponding to the inequality axioms,

$$S_{\leq} = \{x \leq x,\ y \leq x \vee x \leq y,\ y \leq x \vee z \leq y \vee x < z,\ y < x \vee z < y \vee x \leq z\}\ .$$

2.2. Interpollation Axioms: I

1. $\forall x \exists y\ (y < x)$
2. $\forall x \exists y\ (x < y)$
3. $\forall xy\ (x < y \to \exists w\ (x < w < y))$
4. $\forall xyz\ (x < z \wedge y < z \to \exists w\ (x < w < z \wedge y < w < z))$

...

Using, "$\leq$", these can be expanded to include

$$\forall x \; \exists y \; (y \leq x)$$

$$\forall x \; \exists y \; (x \leq y)$$

$$\forall xy \; (x \leq y \rightarrow \exists w \; (x \leq w \leq y))$$

$$\forall xyz \; (x \leq z \wedge y \leq z \rightarrow \exists w \; (x \leq w \leq z \wedge y \leq w \leq z))$$

$$\forall xyz \; (x \leq z \wedge y < z \rightarrow \exists w \; (x \leq w \leq z \wedge y < w < z))$$

...

More precisely, let I, the <u>interpolation axiom</u>, be the infinite set

$$I = \{P. \; \exists \, n \in \mathbf{N} \; \exists \, m \in \mathbf{N} \; \exists \, L$$

(L is a function on $\{0,1,\ldots,n-1\} \times \{0,1,\ldots,n-1\}$

to $\{\leq, <\} \wedge P$ is

$$\forall x_1 \cdots x_n \; \forall y_1 \cdots \forall m \; (\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} (x_i L_{ij} y_j)$$

$$\rightarrow \exists w \, (\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} x_i L_{ij} w \wedge w \, L_{ij} y_j)))\} \, ,$$

where $\mathbf{N} = \{0,1,2,\ldots\}$.

<u>Definition</u>. Let S_I be the (infinite) set of clauses corresponding to I, i.e.,

$$S_I = \{w_{10}(x) < x, \quad w'_{10}(x) \leq x \, , \quad x < w_{01}(x) \, , \quad x \leq w'_{01}(x) \, ,$$

$$x < w_{11}(x,y) \vee y \leq x \, , \quad x \leq w'_{11}(x,y) \vee y < x \, ,$$

$$x < w_{11}(x,y) \vee y \leq x \, , \quad x \leq w'_{11}(x,y) \vee y < x \, ,$$

$$w_{11}(x,y) < y \vee y \leq x \, , \quad w'_{11}(x,y) \leq y \vee y < x \, , \quad \text{(continued)}$$

$$x < w_{21}(x,y,z) \vee z \leq x \vee z \leq y ,$$

$$y < w_{21}(x,y,z) \vee z \leq x \vee z \leq y ,$$

$$w_{21}(x,y,z) < z \vee z \leq x \vee z \leq y ,$$

$$x \leq w'_{21}(x,y,z) \vee z < x \vee z < y ,$$

$$y \leq w'_{21}(x,y,z) \vee z < x \vee z < y ,$$

$$w'_{21}(x,y,z) \leq z \vee z < x \vee z < y ,$$

$$x \leq w''_{21}(x,y,z) \vee z < x \vee z \leq y ,$$

$$\ldots \qquad \}\ .$$

More precisely, let

$$S_I = \{C: \exists\, n \in \mathbf{N}\ \exists\, m \in \mathbf{N}\ \exists\, k \in \mathbf{N}\ \exists\, \ell \in \mathbf{N}\ \exists\, L$$

(L is a function on $\{0,1,\ldots,n-1\} \times \{0,1,\ldots,m-1\}$

to $\{\leq, <\} \wedge k < n \wedge \ell < m \wedge$

$$[C = (\bigvee_{i=1}^{n} \bigvee_{j=1}^{m} \sim (x_i L_{ij} y_j) \vee x_k L_{k\ell} y_\ell)$$

$$\vee C = (\bigvee_{i=1}^{n} \bigvee_{j=1}^{m} \sim (x_i L_{ij} y_j) \vee y_\ell L_{k\ell} x_k))] \ .$$

The axioms for total order plus the interpolation axioms define the theory of dense linear order without endpoints [5]. This theory is decidable [6]. However, the class of formulas we are investigating contains quantification and uninterpreted function symbols and hence is undecidable (since any formula in first order logic can be encoded).

2.3. Equality Axioms

Definition. If S is a set of clauses then S_E is the set of clauses corresponding to the equality axioms for S. (See [8].)

2.4. Axioms for +

1.	$(x+y)+z \leq x+(y+z)$	Associativity
2.	$x+(y+z) \leq (x+y)+z$	Associativity
3.	$x+0 \leq x$	Zero
4.	$x \leq x+0$	Zero
5.	$x+y \leq y+x$	Commutativity
6.	$x+y \leq x+z \rightarrow y \leq z$	Cancellation
7.	$x+y \leq x \rightarrow y \leq 0$	Cancellation
8.	$x+y < x \rightarrow y < 0$	Cancellation

Definition. Let S_+ be the clauses corresponding to the axioms for +,

$$
\begin{aligned}
S_+ = \{ & (x+y)+z \leq x+(y+z) , \\
& x+(y+z) \leq (x+y)+z , \\
& x+y \leq y+x , \\
& x+z < x+y \vee y \leq z , \\
& x+z \leq x+y \vee y < z , \\
& x+0 \leq x , \\
& x \leq x+0 , \\
& x < x+y \vee y \leq 0 , \\
& x \leq x+y \vee y < 0 \} .
\end{aligned}
$$

2.5. Additional Definitions

Definition. Let S be a set of inequality clauses.

A term t is said to be *isolated* in a literal L of S if t occurs in L not within the arguments of any uninterpreted function symbol. t is isolated in S if it is isolated in a literal of S.

Thus t is isolated in each of $t \leq a$, $b < t+c$, $t \leq f(t)$.

A variable x is said to be *eligible* in a clause C (and in S) if it is isolated in C and does not occur within the arguments of an uninstantiated function symbol.

A term t is a *shielding term* of a clause C (and of S) if t has the form

$$f(t_1, \ldots, t_n)$$

where f is an uninstantiated function symbol, and t is isolated and not ground.

For example, x is eligible and $f(y)$ is a shielding term in the clause

$$x+a \leq b \vee f(y) \leq c .$$

t and t' are called *half literals* of the literals $t \leq t'$ and $t < t'$.

Definition. A set S of inequality clauses is said to be:

RC-unsatisfiable if $(S \cup S_{\leq})$ is unsatisfiable, and we write $S \vdash^{RC} \square$.

Definition. If C is an inequality clause of the form

$$\bigvee_{i=1}^{n} (a_i L_i' x) \vee \bigvee_{j=1}^{m} (x L_j'' b_j) \vee E ,$$

where x is a variable which does not occur in E or one of the a_i or b_j, and for each i,j, L_i' is either $\leq$ or $<$, and L_j'' is either $\leq$ or $<$, then

$$R = \bigvee_{i=1}^{n} \bigvee_{j=1}^{n} (a_i L_{ij} b_j) \vee E ,$$

is called a VE-resolvent of C upon x, where L_{ij} is $<$ if both L_i' and L_j'' are $<$, and L_{ij} is $\leq$ otherwise.

Note that x is eligible in C.

Definition. If C is an inequality clause of the form

$$\bigvee_{i=1}^{n} (a_i L_i' x + a_i') \vee \bigvee_{j=1}^{m} (x + b_j' L_j'' b_j) + E ,$$

where x is a variable which does not occur in E or one of the a_i, a_i', b_j or b_j', and for each i,j, $L_i', L_j' \in \{\leq, <\}$, then

$$R = \bigvee_{i=1}^{n} \bigvee_{j=1}^{n} (a_i + b_j' L_{ij} b_j + a_i') \vee E ,$$

is called a VE+ Resolvent of C upon x, where L_{ij} is $<$ if both L_i' and L_j'' are, and $\leq$ otherwise.

Definition. If C_1 and C_2 are inequality clauses of the form

$$C_1 = (a L' b \vee E_1) ,$$
$$C_2 = (b' L'' c \vee E_2) ,$$

where L' and L'' are in $\{\leq, <\}$, and b and b' are unifiable, then

$$R = (A L c \vee E_1 \vee E_2)\sigma$$

is said to be a chain resolvent of C_1 and C_2 upon b and b', where $\sigma = Mgu\{b,b'\}$ and L is $<$ if either of L' or L" is $<$, and $\leq$ otherwise.

Definition. If C is an inequality clause of the form

$$C = (b < b' \vee E)$$

and $\sigma = Mgu\{b,b'\}$, then $E\sigma$ is said to be self-chain resolvent of C upon b and b', $E\sigma$ is also called a chain-resolvent of C.

Definition. If R is a chain resolvent of C_1 and C_2 upon b and b' or a self-chain resolvent of C upon b and b', and

(1) b and b' are both ground, or

(2) b and b' both have the form

$$f(t_1, \ldots, t_n)$$

where f is an uninstantiated function symbol with $n \geq 1$, then R is called an RCF-chain resolvent of C_1 and C_2 upon b and b', (or of C upon b and b').

Definition. If R is an RCF-chain resolvent of C_1 and C_2 upon b and b', and either b or b' is a shielding term then R is called an RCS-chain resolvent of C_1 and C_2 upon b and b', (or of C upon b and b').

Definition. Let C_1 and C_2 be inequality clauses of the form

$$C_1 = (a L' \sum_{i=1}^{n} b_i) \vee E_1 ,$$

$$C_2 = (\sum_{j=1}^{m} b_i' L'' c) \vee E_2 ,$$

where L', $L'' \in \{\leq, <\}$, $k \in \{1,\ldots,n\}$, $\ell \in \{1,\ldots,m\}$, $\sigma = \mathrm{Mgu}\{b_k, b'_\ell\}$, and let

$$R = ((a + \sum_{\substack{j=1 \\ j\neq\ell}}^{m} L\ c + \sum_{\substack{i=1 \\ i\neq k}}^{n}) \vee E_1 \vee E_2)\sigma ,$$

where L is $<$ if both L' and L'' are, and $\leq$ otherwise, and let R' be obtained from R by algebraic simplification whereby like terms on opposite sides of L are cancelled, (if all terms on one side of L are cancelled that side is replaced by 0). Then R' is called an RC+ chain resolvent of C_1 and C_2 upon the literals b_k and b'_ℓ. Also (the self-chaining case) if

$$C = (\sum_{i=1}^{n} a_i L \sum_{j=1}^{m} b_j) \vee E ,$$

where $L \in \{\leq, <\}$, $\sigma = \mathrm{Mgu}\{a_k, b_\ell\}$, then

$$R = ((\sum_{\substack{i=1 \\ i\neq k}}^{n} a_i L \sum_{\substack{j=1 \\ j\neq\ell}}^{m} b_j) \vee E)\sigma ,$$

(algebraically simplified), is called an RC+ chain resolvent of C upon a_k and b_ℓ.

RCF+ and RCS+ chain resolvents are defined similarly, where the appropriate restrictions are maintained on b_k, b_ℓ and a_k.

We note that, in all of these cases, we do not chain-resolve two clauses unless at least one term is cancelled. Thus we would not chain-resolve $a+b \leq c$ and $d+e \leq f$ to get $a+b+d+e \leq c+f$, unless $c=d$, $c=e$, $f=a$, or $f=b$. Also when an intermediate resolvent R is obtained which is simplified to R' by cancelling like terms, we keep only R' and discard R.

Definition. If C is a clause let

$$LE(C) = \begin{cases} '<' & \text{if every literal of } C \text{ has the predicate } '<' , \\ '\leq' & \text{otherwise .} \end{cases}$$

Definition. If C_1 and C_2 are inequality clauses of the form

$$C_1 = (\bigvee_{i=1}^{n} a_i L_i' b_i) \vee E_1 ,$$

$$C_2 = (\bigvee_{j=1}^{m} b_j' L_j'' c_j) \vee E_2 ,$$

where $L_i', L_j'' \in \{\leq, <\}$, $\{b_1, \ldots, b_n, b_1', \ldots, b_m\}$ is unifiable with Mgu σ, then

$$R = ((\bigvee_{i=1}^{n} \bigvee_{j=1}^{n} a_i L_{ij} b_j) \vee E_1 \vee E_2)\sigma$$

is called a multiple cut chain resolvent of C_1 and C_2 upon $b_1, \ldots, b_n$, $b_1', \ldots, b_m'$, where $L_{ij} = LE(L_i, L_j)$. It is also called an RCM-chain resolvent of C_1 and C_2. Also Self-Chain Resolvents are called multiple cut chain resolvents, or RCM-chain resolvents.

RCMF, RCMS, RCMF+, and RCMS+ chain resolvents are defined in a similar way.

Definition. Let C be an inequality clause,

$$C = C' \vee D, \; C' = (a_1 \underset{1}{<} b_1 \vee \ldots \vee a_n \underset{n}{<} b_n), \; n \geq 2 ,$$

where $\underset{i}{<}$ is either $\leq$ or $<$, and let σ be a Mgu of $\{a_1 \leq b_1, \ldots, a_n \leq b_n\}$, with the restriction that

(1) if one of the a_i's is a variable then no b_i can be a variable and σ is a Mgu of $\{b_1, \ldots, b_n\}$, and

(2) if one of the b_i's is a variable then no a_i can be a variable and σ is a Mgu of $\{a_1, \ldots, a_n\}$.

Then $((a_1 \underset{\cdot\cdot}{\leq} b_1) \vee D)\sigma$ is called an RCS-factor of C, where $\underset{\cdot\cdot}{\leq} = LE(C')$.

Thus $(a \leq f(a) \vee g(a) \leq c)$ is an RCS-factor of $(a < f(x) \vee x \leq f(a) \vee g(x) \leq c)$ but not of $(a \leq f(a) \vee x \leq f(a) \vee g(x) \leq c)$. That is, for RCS-factors, we do not allow a variable to unify with a (different) term unless that unification is forced by the unification of other non-variable terms.

Definition. An RC-factor is the same as an RCS-factor, except conditions (1) and (2) are removed.

Definition.

$$FACT(S) = S \cup \{C' : \exists\, C \in S (C' \text{ is an RC-factor of } C)\}.$$

$$FACT\text{-}S(S) = S \cup \{C' : \exists\, C \in S (C' \text{ is an RCS-factor of } C)\}.$$

Definition. If S is a set of inequality clauses, then

$$RC(S) = \{R : \exists\, C_1 \in FACT(S)\ \exists\, C_2 \in FACT(S) \quad (R \text{ is a chain resolvent of } C_1 \text{ and } C_2)\}.$$

$$RC^0(S) = S,$$

$$RC^{n+1}(S) = \cup\, RC(RC^n(S)), \qquad n \in \mathbf{N},$$

$$RC^\infty(S) = \bigcup_{n \in \mathbf{N}} RC^n(S).$$

Definition. If $\square \in RC^{\infty}(S)$ then we write

$$S \vdash^{RC} \square$$

and say that there is an RC-deducting of $\square$ from S (or there is an RC-refutation of S).

Definition. If S is a set of inequality clauses, then

$$VE(S) = \{R: \exists C \in S \ (R \text{ is a VE-Resolvent of } C)\}$$
$$\cup\, S \sim \{C \in S: C \text{ has a VE-Resolvent}\},$$

VE+(S) is defined similarly,

$$RCF(S) = VE(S'), \text{ where}$$
$$S' = \{R: \exists C_1 \in FACT\text{-}S(S) \ \exists C_2 \in FACT\text{-}S(S)$$
$$(R \text{ is a RCF-chain resolvent of } C_1 \text{ and } C_2)\}$$

$$RCS(S) = VE(S'), \text{ where}$$
$$S' = \{R: \exists C_1 \in FACT\text{-}S(S) \ \exists C_2 \in FACT\text{-}S(S)$$
$$(R \text{ is a RCS-chain resolvent of } C_1 \text{ and } C_2)\}$$

etc. for RCF+(S), RCM(S), RCMF(S), RCMS(S), RCMF+(S), and (RMS+(S), except that FACT(S) is used in the definition of RCM(S) (only).

Note that variable elimination is applied immediately to a new resolvent R, when it has an eligible variable, and R is discarded and replaced by its VE-resolvent.

Definition.

$$RCF^0(S) = S ,$$

$$RCF^{n+1}(S) = RCF^n(S) \cup RCF(RCF^n(S)) ,$$

$$RCF^\infty(S) = \bigcup_{n \in \mathbb{N}} RCF^n(S) .$$

Similarly for $RCS^\infty(S), \ldots, RCMS+^\infty(S)$.

Definition. If $\square \in RCF^\infty(S)$ we write

$$S \vdash_{RCF} \square$$

and say that there is an RCF-deduction of $\square$ from S. Similarly for

$$S \vdash_{RCF} \square$$

$$\vdots$$

$$S \vdash_{RCMS+} \square .$$

3. Completeness Results

3.1. RCF Completeness

Lemma 1. If S is a set of inequality clauses, σS is ground, S is not ground, and S has no eligible variables, then S contains a shielding term t for which $t\sigma \neq x\sigma$ for all isolated variables x in S.

Proof. If S has no isolated variable we are finished. So let

x_1 be an isolated variable in clause C_1,
$f_1(x_1)$ be a shielding term in C_1 (since x_1 is not eligible, by hypothesis).

Now if $f_1(x_1)\sigma \neq V\sigma$ for each isolated variable V in S, we are finished. So suppose that

$f_1(x_1)\sigma = x_2\sigma$ for some isolated variable in clause C_2,
$f_2(x_2)$ is a shielding term in C_2,
...
x_n is an isolated variable in clause C_n
$f_{n-1}(x_{n-1})\sigma = x_n\sigma$,
$f_n(x_n)$ is a shielding term in C_n
....

If this were the case then we would have

$$f_1(x_1)/x_2\ ,\quad f_2(x_2)/x_3\ ,\ \ldots,\ f_n(x_n)/x_{n+1}\ ,\ \ldots$$

or

$$f_n f_{n-1} f_{n-2} \cdots f_2 f_1(x_1)/x_{n+1}\ .$$

But σ has finite depth, so this process has to terminate. It can only terminate if one of the x_i is eligible, or if one of the $f_i(x_i)$ is such that

$$f_i(x_i)\sigma \neq x\sigma$$

for any isolated variable x in S.

Q.E.D.

Lemma 2. If S is an RC-unsatisfiable set of ground clauses, and c is a half literal of S (i.e., $c \leq d$, $d \leq c$, $c < d$, or $d < c$ is in S, for some d), then there is an RC-refutation $\mathcal{D}$ of S for which any chaining on terms other than c is done on clauses not containing c (as a half literal).

(That is, all chainings on c are done first, and then only clauses not containing c are retained for the remainder of the refutation.)

Proof. The proof is by induction on the excess literal parameter $k(S)$.*

Case 1. $k(S) = -1$. Then $\square \in S$ and we are finished.

Case 2. $k(S) = 0$, $\square \notin S$.

In this case the clauses of S are all units and by Lemma 2, Appendix I, S contains a sequence of unit clauses

$$a_1 \stackrel{\cdot}{\leq} a_2 : a_2 \stackrel{\cdot}{\leq} a_3 : \ldots : a_{n-1} \stackrel{\cdot}{\leq} a_n : a_n \stackrel{\cdot}{\leq} a_1 ,$$

*The excess literal parameter $k(S)$ is defined as

$$k(S) = \left(\sum_{C \in S} |C|\right) - |S| .$$

That is $k(S)$ is the total number of occurrances of literals minus the number of clauses in S.

where each $\lesssim$ is either $\leq$ or $<$ and at least one of the $\lesssim$ is $<$.

If any of the a_i are c's, then they can be chained upon first..

Case 3. (Induction Step)

Suppose $k(S) = n$, $n \geq 1$, and that for each set S' of ground clauses which is RC-unsatisfiable and for which $k(S') < n$, there is an RC-refutation $\mathcal{D}'$ of S' for which any chaining on a term other than c is done on clauses not containing c (as a half literal).

Then S has at least one non-unit clause C (since $k(S) > 0$). Let

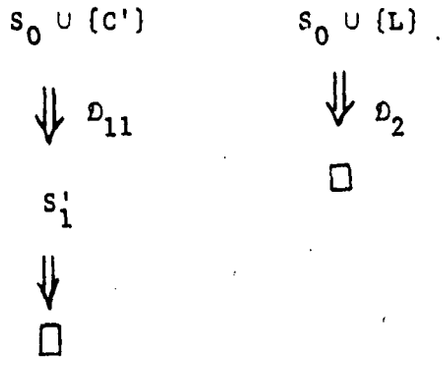
$$C = C' \vee L$$

where C' is a clause and L is a unit clause. Let

$$S_0 = S \sim \{C\},$$
$$S_1 = S_0 \cup \{C'\}, \quad S_2 = S_0 \cup \{L\}.$$

Then S_1 and S_2 sub some S and hence are RC-unsatisfiable. Also $k(S_1) < n$, $k(S_2) < n$, and hence by the induction hypothesis, there are RC-refutations $\mathcal{D}_1$ and $\mathcal{D}_2$ of S_1 and S_2, respectively, for which any chaining on terms other than c is done on clauses not contain c.

Let $\mathcal{D}_{11}$ be the first part of $\mathcal{D}_1$ in which chaining is done only on c, and $\mathcal{D}_{12}$ be the rest of $\mathcal{D}_1$ (the last part of $\mathcal{D}_1$). And let S_1' be a set of resolutents produced by $\mathcal{D}_{11}$ which do not contain c (as a half literal), but such that $\mathcal{D}_{12}$ produces $\square$ from S_1'.



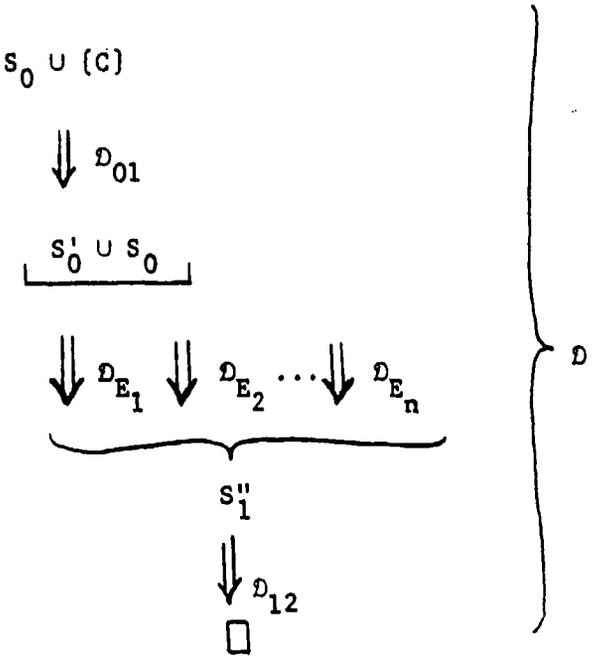
Now build $\mathfrak{D}$ out of $\mathfrak{D}_{11}$, $\mathfrak{D}_2$, and $\mathfrak{D}_{12}$ as follows:

Let $\mathfrak{D}_{01}$ be the same as $\mathfrak{D}_{11}$ except that C' is replaced by C (and some descendents of C' have the additional literal L), and let S_0' be produced by $\mathfrak{D}_{01}$ from S (similarly as S_1' is produced by $\mathfrak{D}_{11}$ from S_1).

For each clause E in S_1', we have by Lemma 1, Appendix I, that either E or $(E \vee L)$ is in S_0'. For each such $(E \vee L)$ in S_0', let $\mathfrak{D}_E$ be the same as $\mathfrak{D}_2$ except that L is replaced by $(E \vee L)$ and some descendents of $(E \vee L)$ have additional literals from E. Thus $\mathfrak{D}_E$ when applied to $S_0 \cup \{E \vee L\}$ will produce a clause E' which subsumes E. (By Lemma 1, Appendix I).

By applying such a deducting $\mathfrak{D}_E$ to each such $(E \vee L)$ in S_0', we obtain from $(S_0' \cup S_0)$ a set S_1'' of clauses which subsumes S_1'. And then we apply $\mathfrak{D}_{12}$ to S_1'' to obtain □.

$\mathfrak{D}$ is made up of $\mathfrak{D}_{01}$, several of the $\mathfrak{D}_E$'s, and $\mathfrak{D}_{12}$.



Since $\mathfrak{D}_{01}$ consists of chainings only on c, since the first part of $\mathfrak{D}_{E_i}$ consists of chainings only on c for each i, since the $\mathfrak{D}_{E_i}$ are done in parallel, and since $\mathfrak{D}_{12}$ chains only on clauses not containing c, it follows that $\mathfrak{D}$ has the desired properties.

Q.E.D.

A different proof of Lemma 2, due to Ken Kunen, is given in Appendix II.

Lemma 3. If S is an RC-unsatisfiable set of clauses (S may contain more than one variant of a particular clause), $S\sigma$ is ground and RC-unsatisfiable, t is a half literal of S,

$$\mathscr{C} = \{t' : t' \text{ is a half literal of } S \text{ and } t'\sigma = t\sigma\},$$

then there is an RC-deduction $\mathfrak{D}'$ of a set S' from S for which

(1) each step in $\mathfrak{D}'$ is a chaining on a member of $\mathscr{C}$,

(2) S' contains no member of $\mathscr{C}$ as a half literal,

(3) $S'\sigma$ (and therefore S') is RC-unsatisfiable.

Proof. Apply Lemma 2 to $S\sigma$, with $t\sigma$ for c, to obtain an RC-refutation $\mathcal{D}''$ of $S\sigma$ for which any chaining on terms other than $t\sigma$ is done on clauses not containing $t\sigma$ (as a half literal).

Let S'' be the clauses obtained by $\mathcal{D}''$ on $S\sigma$ where only chainings on $t\sigma$ are done, and let S_0' be those clauses of $S'' \cup S\sigma$ not containing $t\sigma$ (as a half literal). Since any chaining on terms other than $t\sigma$ is done on clauses not containing $t\sigma$, it follows that $\mathcal{D}''$ is an RC-refutation of S_0'.

$\mathcal{D}'$ is obtained from $\mathcal{D}''$ and S' from S_0' by lifting. Conclusions (1), (2) and (3) follow immediately.

Lemma 4. (RC-completeness Theorem)

If S is an RC-unsatisfiable set of clauses then there is an RC-deduction of $\square$ from S.

Proof. Let S' be an RC-unsatisfiable set of ground instances of S. Then by Lemma 2 there is an RC-refutation $\mathcal{D}$ of S'. Lifting $\mathcal{D}$ gives the desired conclusion.

Remark. The deductions provided by Lemmas 2 and 4 may employ tautologies, as the following example shows.

Example

1.	$b \leq a \quad c \leq a \quad d \leq a$	S
2.	$a < b \quad c < c \quad a < d$	
3.	$c \leq b$	
4.	$b \leq c$	
5.	$d \leq b$	
6.	$b \leq d$	

Notice that each chaining on S results in a tautology. To show that S is RC-unsatisfiable, the following deduction (using tautologies) is given.

7.	$c \leq a \quad d \leq a \quad a < c \quad a < d$	1,2
8.	$c \leq a \quad d \leq a \quad b < c \quad a < d$	1,7
9.	$c \leq a \quad d \leq a \quad b < c \quad b < d$	1,8
10.	$c < b \quad d \leq a \quad b < c \quad b < d \quad a < c \quad a < d$	9,2
11.	$c < b \quad d \leq a \quad b < c \quad b < d \quad a < d$	9,10
12.	$c < d \quad d \leq a \quad b < c \quad b < d \quad c < d$	9,11
13.	$c < b \quad d < b \quad b < c \quad b < d \quad c < d \quad a < c \quad a < d$	12,2
14.	$c < b \quad d < b \quad b < c \quad b < d \quad c < d \quad d < c \quad a < d$	12,13
15.	$c < b \quad d < b \quad b < c \quad b < d \quad c < d \quad d < c$	12,14
16.	$c \leq d$	3,6
17.	$d \leq c$	5,4
18.	$\square$	15,4,6,3,5,17,16

The use of tautologies in RC proofs can be avoided if we use "multiple cuts" whereby for example clauses 1 and 2 above produce in one step the clause 15, and intermediate clauses 7-14 are not produced or retained. See [9].

Lemma 5. If S is an RC-unsatisfiable set of clauses, $S\sigma$ is ground and RC-unsatisfiable, $C \in S$, x is a variable,

$$C = (\bigvee_{i=1}^{n} x < a_i \vee \bigvee_{j=1}^{m} b_j < x \vee E)$$

where x does not occur in a_i, b_j or E, then

$$S' = S \sim \{C\} \cup \{ \bigvee_{i=1}^{n} \bigvee_{j=1}^{n} b_j < a_i \vee E \}$$

is RC-unsatisfiable, and $S'\sigma$ is RC-unsatisfiable. Also the shielding terms of S' are those of S. (A similar theorem holds when some or all of the '$<$' in C are replaced by '$\leq$', and appropriate changes are made in S'.)

Proof. Let

$$C' = (\bigvee_{i=1}^{n} \bigvee_{j=1}^{m} b_j < a_i \vee E) ,$$

$$S_0 = S \sim \{C\} .$$

We must show that $(S_0 \cup \{C'\})\sigma$ is unsatisfiable. We will show that any model for $(S_0 \cup \{C'\})\sigma$ is a model for $S\sigma = (S_0 \cup \{C\})\sigma$.

Suppose M is a model for $(S_0 \cup \{C'\})\sigma$. If M is a model for $E\sigma$ then M is a model for $C\sigma$ and we are through. Otherwise M is a model for $(b_j\sigma < a_i\sigma)$, for some i, j.

If M is already defined on $(x\sigma < a_i\sigma)$ and $(b_j\sigma < x\sigma)$, then, since $(b_j\sigma < a_i\sigma)$ is TRUE under M, it follows that either $(x\sigma < a_i\sigma)$ or $(b_j\sigma < x\sigma)$ is TRUE under M. If M is not defined on these two literals, we arbitrarily define it to be TRUE on the first and FALSE on the second (or vice versa). In either case M is a model for $C\sigma$ and is therefore a model for $S\sigma$.

Clearly the shielding terms of S' are those of S.

Q.E.D.

Lemma 6. If S is an RC-unsatisfiable set of clauses then there exists a set S_1 of variants of S and a substitution σ such that $S_1\sigma$ is ground and RC-unsatisfiable.

Theorem 1. If S is an RC-unsatisfiable set of clauses then there is an RCF-refutation of S.

Proof. By Lemma 6 there is a set S_1 of variants of S and a substitution σ for which $S_1\sigma$ is ground and RC-unsatisfiable. WLOG assume that S has no eligible variable.

Recursively define $S_2, S_3, \ldots$ as follows:

If S_i is ground, halt.

If S_i is ground, halt.

If S_i is not ground, use Lemma 1 to select a shielding term t from S_i for which $\sigma t \neq \sigma x$ for any isolated variable x in S_i, and let

$$\mathfrak{G} = \{t' : t'\sigma = t\sigma \wedge t' \text{ is a half literal of } S_i\},$$

and use Lemma 3 to obtain an RC-deduction $\mathfrak{D}_i$ of a set S'_{i+1} from S_i for which each step in $\mathfrak{D}_i$ is a chaining on a member of $\mathfrak{G}$, S'_{i+1} contains no member of $\mathfrak{G}$, (as a half literal), and S'_{i+1} and $S'_{i+1}\sigma$ are RC-unsatisfiable. Let $S_{i+1} = VE(S'_{i+1})$.

We observe that variable elimination (i.e., the use of Lemma 4) on a set S' does not increase the number of half literals in $S'\sigma$. Furthermore, in applying Lemma 3, the half literals of S_{i+1} are a subset of those of S_i, and $t\sigma$ is a half literal of $S_i\sigma$ but not $S_{i+1}\sigma$. So the use of Lemma 3 steadily decreases the number of half literals in $S_i\sigma$. Therefore the sequence, $S_1, S_2, \ldots$, must terminate in an RC-unsatisfiable ground set S_n. Let $\mathfrak{D}_G$ be the RCF-refutation of S_n.

Since the shielding term chosen by Lemma 1 is such that

$$t\sigma \neq x\sigma$$

for any variable x, it follows that if $t\sigma = t'\sigma$, then t and t' have the form

$$f(t_1, \ldots, t_n)$$

where f is an uninstantiated function symbol, and therefore each member of $\mathfrak{G}$ has this form. And since $\mathfrak{D}_i$ chains only on members of $\mathfrak{G}$ it follows that each of the steps of $\mathfrak{D}_i$ produces an RCF-resolvent.

Since variable elimination steps are also RCF-steps it would appear that $\mathfrak{D}_i$ and $\mathfrak{D}_i'$ together form a RCF-deduction of S_{i+1} from S_i. But in the definition of $RCF^n(S)$ we required that variable elimination be applied on a resolvent immediately when it is produced (if it has an eligible variable), so we cannot follow $\mathfrak{D}_i$ by $\mathfrak{D}_i'$, but must intermingle the two, by reording the VE and RCF steps. In particular, by [11], there is an RCF-deduction $\mathfrak{D}_i''$ of S_{i+1} from S_i, for each i, $i = 1, n-1$.

And by putting together the deductions

$$\mathfrak{D}_1'', \mathfrak{D}_2'', \ldots, \mathfrak{D}_{n-1}'', \mathfrak{D}_G,$$

we obtain an RCF-refutation of S.

Q.E.D.

Theorem 2. (RCF Completeness Theorem)

Let

S be a set of inequality clauses,

$S_{\leq}$ be the set of clauses for the inequality axioms,

S_I be the set of clauses for the interpolation axioms,

and suppose $(S \cup S_{\leq} \cup S_I)$ is unsatisfiable. Then there is an RCF-deduction of $\square$ from S.

Proof. By definition $(S \cup S_I)$ is RC-unsatisfiable. Thus by Theorem 1 there is an RCF-deduction $\mathcal{D}$ of $\square$ from $(S \cup S_I)$. But no clause of S_I can be a part of a (productive) step in $\mathcal{D}$, so $\mathcal{D}$ is an RCF-deduction of $\square$ from S.

To see why a clause of S_I cannot be part of a (productive) step in $\mathcal{D}$, recall that S_I is the set of clauses

$$x_k \leq w_{nm}(x_1, \ldots, x_n, y_1, \ldots, y_m) \vee \bigvee_{i=1}^{n} \bigvee_{j=1}^{n} (y_j < x_i)$$

$$w_{nm}(x_1, \ldots, x_n, y_1, \ldots, y_m) \leq y_\ell \vee \bigvee_{i=1}^{n} \bigvee_{j=1}^{n} (y_j < x_i)$$

$$k = 1,n; \quad \ell = 1, m; \quad n \geq 0; \quad m \geq 0,$$

together with similar clauses when $\leq$ and $<$ are interchanged.

Consider the case when $n = 1$, $m = 1$.

$$CI_1 = (x \leq w(x,y) \vee y < x)$$

$$CI_2 = (w(x,y) \leq y \vee y < x)$$

(we have dropped the subscript on w). Since the symbol 'w' occurs only in CI_1 and CI_2 and nowhere else in S, it follows that no chaining on $w(x,y)$ with another clause in S is allowed in $\mathcal{D}$, because it would have to match a variable. And chaining CI_1 with CI_2 would produce the tautology

$$x \leq y \vee y < x$$

which again cannot be used in any step of D since matching on variables is forbidden. Hence CI_1 and CI_2 are not used in a productive way in D and can be removed from $S \cup S_I$. Similarly other members of S_I can be removed.

Q.E.D.

Lemma 7. If

S is a set of inequality and equality clauses,

$S_{\leq}$ is the set of clauses for the inequality axioms,

S'' is obtained from S by replacing each literal of the form $(a = b)$ by $(a \leq b \wedge b \leq a)$ and reclausing if necessary,

and S is unsatisfiable, then $(S'' \cup S_{\leq})$ is RC-unsatisfiable.

Proof. The following is a partial sketch of the proof for the ground case. Lifting gives the general case.

Suppose two clauses

$$C_1 = (a = b \vee E_1)$$
$$C_2 = (a \neq b \vee E_2)$$

in S are resolved to obtain

$$R = (E_1 \vee E_2) .$$

If C_1 and C_2 have no other "$=$" symbol then C_1 is converted to the two clauses in S'',

$$C_{1.1} = (a \leq b \vee E_1)$$
$$C_{1.2} = (b \leq a \vee E_1)$$

and C_2 is converted to

$$C_2' = (a < b \vee b < a \vee E_2) .$$

RC-chaining $C_{1.1}$ and $C_{1.2}$ with C_2' gives R.

Theorem 3. Let

S be a set of inequality and equality clauses,

$S_{\leq}$ be the set of clauses for the inequality axioms,

S_E be the set of clauses for the equality axioms for the sets S,

S_I be the set of clauses for the interpolation axioms,

S' be obtained from $S \cup S_E$ by replacing each literal $a = b$ by $(a \leq b \wedge b \leq a)$ and reclausing if necessary,

and suppose $(S \cup S_{\leq} \cup S_I)$ is E-unsatisfiable, and $S \cap S_I = \emptyset$. Then there is an RCF-deduction of $\square$ from S'.

Proof. In this proof we use the following notation: For any set U of inequality and equality clauses,

U_E is the set of clauses for the equality axioms for U,

U" is obtained from U by replacing each literal of the form $a = b$ by $(a \leq b \wedge b \leq a)$ and reclausing if necessary.

Thus, in the above, $S' = S'' \cup S_E''$, and we must show that there is an RCF-deduction of $\square$ from $S'' \cup S_E''$.

We first give an outline of the proof:

$S \cup S_{\leq} \cup S_I$ is E-unsatisfiable (Hypothesis of Theorem 3)

↓ Reference [8]

$S \cup S_{\leq} \cup S_I \cup S_E \cup S_{IE}$ is unsatisfiable

↓ Lemma 7

$S'' \cup S_{\leq} \cup S_I \cup S''_E \cup S''_{IE}$ is RC-unsatisfiable

(Note $S''_{\leq} = S_{\leq}$, $S''_I = S_I$)

↓ Theorem 2 (with $S'' \cup S''_E \cup S''_{IE}$ for S)

There is an RCF-deduction of $\square$ from $(S'' \cup S''_E \cup S''_{IE})$

↓ See below.

There is an RCF-deduction of $\square$ from $(S'' \cup S''_E)$.

The last step follows because if $\mathcal{D}$ is an RCF-deduction of $\square$ from $S'' \cup S''_E \cup S''_{IE}$ then we can omit from $\mathcal{D}$ those steps involving S''_{IE}. Because S''_{IE} has only clauses of the form

$$C_{I1} = x_1 < x'_1 \vee x'_1 < x_1 \vee \ldots \vee y_m < y'_m \vee y_m < y_m \vee w_{nm}(x_1, \ldots, x_n, y_1, \ldots, y_m) \leq w_{nm}(x'_1, \ldots, x'_n, y'_1, \ldots, y'_m) ,$$

(and similar clauses, see Section 2), and since the symbol "w_{nm}" does not occur in $S'' \cup S''_E$, no RCF step can use C_I unless C_I is chained with itself.

But such a chaining only produces a RCF-resolvent

$$x_1 < x_1' \vee \ldots \vee y_m < y_m' \vee w_{rm}(x_1, \ldots, y_m) \leq w_{rm}(x_1, \ldots, y_m)$$

which can again only be used against members of $RCF^{\infty}(S''_{IE})$. So no interaction with $S'' \cup S''_E$ is possible.

3.2. RCF+ Completeness

Lemma 9 (Ground unit RC+ Completeness). If S is an RC+ unsatisfiable set of ground unit clauses, then there is an RC+ deduction of $\square$ from S.

This follows essentially from a consistency criterion used in linear programming. See [10]. Also see Lemma 3, Appendix I.

Lemma 10. If S is an RC+ unsatisfiable set of ground unit clauses, and c is an isolated term* of S, then there is an RC+ refutation D of S for which any chaining on terms other than c is done on clauses not containing c (as an isolated term).

Proof. Use Lemma 9.

Lemma 11. (Like Lemma 2) If S is an RC+ unsatisfiable set of ground clauses, and c is an isolated term of S, then there is an RC+ refutation D of S for which any chaining on terms other than c is done on clauses not containing c (as an isolated term).

Proof. The proof is by induction on the excess literal parameter k(S).

Case 1. $k(S) = -1$. Then $\square \in S$.

Case 2. $k(S) = 0$, $\square \notin S$.

In this case the clauses of S are ground unit clauses, and the desired result follows from Lemma 10.

Case 3. (Induction Step) The proof of this case follows exactly as the proof of Case 3 in Lemma 2, except the expression "half literal" is replaced by "isolated term".

---

*Recall that a term is isolated if it occurs not within the arguments of any uninstantiated function symbol. E.g., $t \leq a$, $t + a \leq b$, $a + t + b < c$, etc.

Lemma 12. (Like Lemma 3) If S is an RC+ unsatisfiable set of clauses, $S\sigma$ is ground and RC+ unsatisfiable, t is an isolated term of S,

$$\mathfrak{T} = \{t' : t \text{ is an isolated term of } S \text{ and } t'\sigma = t\sigma\},$$

then there is an RC+ deduction $\mathcal{D}'$ of a set S' from S for which

(1) each step in $\mathcal{D}'$ is a chaining on a member of $\mathfrak{T}$,

(2) S' contains no member of $\mathfrak{T}$ (as an isolated term),

(3) $S'\sigma$ (and therefore S) is RC+ unsatisfiable.

Proof. Similar to that of Lemma 3.

Lemma 13. (Like Lemma 5) If S is an RC+ unsatisfiable set of clauses, $C \in S$, x is an eligible variable in C, and R is a VE+ Resolvent of C upon x, then $S \sim \{C\} \cup \{R\}$ is RC+ unsatisfiable.

Proof. The proof is similar to that of Lemma 5.

Theorem 4. If S is an RC+ unsatisfiable set of clauses then there is an RCF+ refutation of S.

Proof. Very much like that of Theorem 1.

APPENDIX Theorem Prover Listing

The following is an excerpt from the Interlisp implementation of the experimental theorem prover developed during the second year of the project. The excerpt exhibits the main procedures in part of the theorem prover that reduces propositional structure.

```
(PROVE
  (LAMBDA (FORM)
    (NEW.CONTEXT (AND.SIMP (LIST FORM)))))

(NEW.CONTEXT
  (NLAMBDA (X)
    (PROG ((SIGNATURE.ALIST SIGNATURE.ALIST)
           (FIND.PTR.ALIST FIND.PTR.ALIST)
           (USE.ALIST USE.ALIST)
           (INEQLIST (APPEND INEQLIST))
           (IF.ALIST IF.ALIST))
          (RETURN (EVAL X)))))

(AND.SIMP
  (LAMBDA (STACK SUBGOALS FAST.FLG)                    (* edited:
                                                       "19-Feb-81 21:02")
    (PROG ((DEFER.POT (CONS NIL (AND SUBGOALS (APPEND SUBGOALS))))
           EXP SINGLE ADD.ELEM NOT.EXP)
      TOP (while STACK
             do
              ((SETQ EXP (CAR STACK))
               (COND
                 ((ATOM EXP)
                   (SELECTQ EXP
                            (TRUE (SETQ STACK (CDR STACK)))
                            (FALSE (RETFROM (QUOTE NEW.CONTEXT)
                                            (QUOTE FALSE)))
                            (PROG2 (OR (ADD.EQ (LIST (QUOTE EQUAL)
                                                     (QUOTE TRUE)
                                                     EXP))
                                       (SETQ SINGLE
                                         (COND
                                           (SINGLE (QUOTE FALSE))
                                           (T EXP))))
                                   (SETQ STACK (CDR STACK)))))
                 (T
                   (SELECTQ
                     (CAR EXP)
                     (NOT
                       (SETQ NOT.EXP (CADR EXP))
                       (COND
                         ((ATOM NOT.EXP)
                           (SELECTQ
                             NOT.EXP
                             (TRUE (RETFROM (QUOTE NEW.CONTEXT)
```

```
                    (QUOTE FALSE)))
      (FALSE (SETQ STACK (CDR STACK)))
      (PROG2 (OR (ADD.EQ (LIST (QUOTE EQUAL)
                               (QUOTE FALSE)
                               NOT.EXP))
                 (SETQ SINGLE
                   (COND
                     (SINGLE (QUOTE FALSE))
                     (T EXP))))
             (SETQ STACK (CDR STACK)))))
(T
  (SELECTQ
    (CAR NOT.EXP)
    (NOT (SETQ STACK (CONS (CADR NOT.EXP)
                           (CDR STACK))))
    (AND
      (COND
        ((CDR NOT.EXP)
          (RPLACD
            DEFER.POT
            (CONS
              (CONS
                (QUOTE OR)
                (for ARG in (CDR NOT.EXP)
                   collect
                    (LIST (QUOTE NOT)
                          ARG)))
              (CDR DEFER.POT)))
          (SETQ STACK (CDR STACK)))
        (T (RETFROM (QUOTE NEW.CONTEXT)
                    (QUOTE FALSE)))))
    (OR
      (COND
        ((CDR NOT.EXP)
          (SETQ STACK
            (NCONC (for ARG
                      in (CDR NOT.EXP)
                      collect
                       (LIST (QUOTE NOT)
                             ARG))
                   (CDR STACK))))
        (T (SETQ STACK (CDR STACK)))))
    (IMPLIES
      (SETQ STACK
        (CONS (CADR NOT.EXP)
              (CONS (LIST (QUOTE NOT)
                          (CADDR NOT.EXP))
                    (CDR STACK)))))
    (IF
      (RPLACD
        DEFER.POT
        (CONS (LIST (QUOTE IF)
                    (CADR NOT.EXP)
                    (LIST (QUOTE NOT)
```

```
                                    (CADDR NOT.EXP))
                                (LIST (QUOTE NOT)
                                    (CADDDR NOT.EXP)))
                            (CDR DEFER.POT)))
                    (SETQ STACK (CDR STACK)))
                (IF.OBJ (SETQ IF.ALIST
                            (CONS (CADR NOT.EXP)
                                IF.ALIST))
                        (SETQ STACK
                            (CONS (LIST (QUOTE NOT)
                                    (CADDR NOT.EXP))
                                (CDR STACK))))
                (IFF
                    (RPLACD
                        DEFER.POT
                        (CONS
                            (LIST (QUOTE OR)
                                (LIST (QUOTE AND)
                                    (CADR NOT.EXP)
                                    (LIST (QUOTE NOT)
                                        (CADDR NOT.EXP)))
                                (LIST (QUOTE AND)
                                    (CADDR NOT.EXP)
                                    (LIST (QUOTE NOT)
                                        (CADR NOT.EXP))))
                            (CDR DEFER.POT)))
                    (SETQ STACK (CDR STACK)))
                (SELECTQ (SETQ ADD.ELEM (ADD.ELEM.REL
                                NOT.EXP T))
                        (NIL (SETQ STACK (CDR STACK))
                            (SETQ SINGLE
                                (COND
                                    (SINGLE (QUOTE FALSE))
                                    (T EXP))))
                        (T (SETQ STACK (CDR STACK)))
                        (SETQ STACK (CONS ADD.ELEM
                                        (CDR STACK))))))
        ))
    (AND (SETQ STACK (APPEND (CDR EXP)
                            (CDR STACK))))
    (OR (COND
            ((CDR EXP)
                (RPLACD DEFER.POT (CONS EXP (CDR DEFER.POT)
                                        )))
            (T (RETFROM (QUOTE NEW.CONTEXT)
                        (QUOTE FALSE))))
        (SETQ STACK (CDR STACK)))
    (IMPLIES (RPLACD
                DEFER.POT
                (CONS (LIST (QUOTE OR)
                            (LIST (QUOTE NOT)
                                (CADR EXP))
                            (CADDR EXP))
                    (CDR DEFER.POT)))
```

```
                        (SETQ STACK (CDR STACK)))
                    (IF (RPLACD DEFER.POT (CONS EXP (CDR DEFER.POT)))
                        (SETQ STACK (CDR STACK)))
                    (IF.OBJ (SETQ IF.ALIST (CONS (CADR EXP)
                                                 IF.ALIST))
                            (SETQ STACK (CONS (CADDR EXP)
                                              (CDR STACK))))
                    (IFF (SETQ STACK
                               (CONS (LIST (QUOTE AND)
                                           (LIST (QUOTE IMPLIES)
                                                 (CADR EXP)
                                                 (CADDR EXP))
                                           (LIST (QUOTE IMPLIES)
                                                 (CADDR EXP)
                                                 (CADR EXP)))
                                     (CDR STACK))))
                    (SELECTQ (SETQ ADD.ELEM (ADD.ELEM.REL EXP))
                             (NIL (SETQ SINGLE (COND
                                               (SINGLE (QUOTE FALSE))
                                               (T EXP)))
                                  (SETQ STACK (CDR STACK)))
                             (T (SETQ STACK (CDR STACK)))
                             (SETQ STACK (CONS ADD.ELEM (CDR STACK))))))
                ))))
            (COND
              ((CDR DEFER.POT)
                (FAST.ITERATE DEFER.POT)
                (COND
                  (STACK (GO TOP))
                  (FAST.FLG)
                  ((CDR DEFER.POT)
                    (SLOW.ITERATE DEFER.POT)
                    (COND
                      (STACK (GO TOP))
                      ((CDDR DEFER.POT)
                        (SPLIT.RECURSE (CDR DEFER.POT))
                        (RPLACD DEFER.POT)
                        (AND STACK (GO TOP))))))))))
            (RETURN (COND
                      (SINGLE (COND
                                ((OR (EQ SINGLE (QUOTE FALSE))
                                     (CDR DEFER.POT))
                                  NIL)
                                (T SINGLE)))
                      ((CDR DEFER.POT)
                        (COND
                          ((CDDR DEFER.POT)
                            NIL)
                          (T (CADR DEFER.POT))))
                      (T (QUOTE TRUE)))))))

(FAST.ITERATE
  (LAMBDA (DEFER.POT.PTR)
    (while (CDR DEFER.POT.PTR) bind SIMP
```

```
      do (SELECTQ (SETQ SIMP (SELECTQ (CAADR DEFER.POT.PTR)
                                      (OR (OR.SIMP (CADR DEFER.POT.PTR)
                                                   NIL T))
                                      (IF (IF.SIMP (CADR DEFER.POT.PTR)
                                                   ))
                                      NIL))
                  (TRUE (RPLACD DEFER.POT.PTR (CDDR DEFER.POT.PTR)))
                  (FALSE (RETFROM (QUOTE NEW.CONTEXT)
                                  (QUOTE FALSE)))
                  (NIL (SETQ DEFER.POT.PTR (CDR DEFER.POT.PTR)))
                  (PROGN (SETQ STACK (CONS SIMP STACK))
                         (RPLACD DEFER.POT.PTR (CDDR DEFER.POT.PTR)))))
   ))

(FAST.PROVE
  (LAMBDA (FORM)
    (NEW.CONTEXT (AND.SIMP (LIST FORM)
                           NIL T))))

(OR.SIMP
  (LAMBDA (STACK SUBGOALS FAST.FLG)
    (PROG (SIMP)
          (SETQ STACK (for X in (CDR STACK) collect
                                            (LIST (QUOTE NOT)
                                                  X)))
          (RETURN (SELECTQ (SETQ SIMP (NEW.CONTEXT (AND.SIMP STACK
                                                              SUBGOALS
                                                              FAST.FLG)))
                           (TRUE (QUOTE FALSE))
                           (FALSE (QUOTE TRUE))
                           (NIL NIL)
                           (LIST (QUOTE NOT)
                                 SIMP))))))

(SLOW.ITERATE
  (LAMBDA (DEFER.POT.PTR)
    (while (CDR DEFER.POT.PTR) bind SIMP
       do
        (SELECTQ
          (SETQ SIMP
            (SELECTQ (CAADR DEFER.POT.PTR)
                     (OR (OR.SIMP (CADR DEFER.POT.PTR)))
                     (IF (OR.SIMP (CAR (RPLACA (CDR DEFER.POT.PTR)
                                               (CONVERT.IF.TO.OR
                                                 (CADR DEFER.POT.PTR))))
                                  ))
                     NIL))
          (TRUE (RPLACD DEFER.POT.PTR (CDDR DEFER.POT.PTR)))
          (FALSE (RETFROM (QUOTE NEW.CONTEXT)
                          (QUOTE FALSE)))
          (NIL (SETQ DEFER.POT.PTR (CDR DEFER.POT.PTR)))
          (PROGN (SETQ STACK (CONS SIMP STACK))
                 (RPLACD DEFER.POT.PTR (CDDR DEFER.POT.PTR)))))))
```

```
(SPLIT.RECURSE
  (LAMBDA (GOALS)
    (PROG (SINGLE SIMP)
          (RETURN (SELECTQ (for DISJUNCT in (CDAR GOALS)
                              do (SELECTQ (SETQ SIMP
                                                (NEW.CONTEXT
                                                  (AND.SIMP (LIST DISJUNCT)
                                                            (CDR GOALS))))
                                          (NIL (RETURN (QUOTE NO.LUCK)))
                                          (FALSE)
                                          (TRUE (RETURN (QUOTE TRUE)))
                                          (COND
                                            (SINGLE
                                              (RETURN (QUOTE NO.LUCK)))
                                            (T (SETQ SINGLE SIMP)))))
                           (NO.LUCK (RETFROM (QUOTE NEW.CONTEXT)))
                           (TRUE)
                           (COND
                             (SINGLE (SETQ STACK (LIST SINGLE)))
                             (T (RETFROM (QUOTE NEW.CONTEXT)
                                         (QUOTE FALSE)))))))))
```